การสังเคราะห์ข้อกำหนดรูปนัยโดยใช้เครือข่ายอนุภาคความต้องการ

นายวิวัฒน์ วัฒนาวุฒิ

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรดุษฎีบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2545

ISBN 974-17-1532-3

ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

FORMAL SPECIFICATION SYNTHESIS
USING REQUIREMENTS PARTICLE NETWORKS

Mr. Wiwat Vatanawood

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

A Dissertation Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy in Computer Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic year 2002
ISBN 974-17-1532-3

Thesis Title	Formal Specification Synthesis using Requirements Particle Networks
By	Mr. Wiwat Vatanawood
Field of Study	Computer Engineering
Thesis Advisor	Associate Professor Wanchai Rivepiboon, Ph.D.

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Doctor's Degree

…………………………………….. Dean of Faculty of Engineering
(Professor Somsak Punyakeow, D.Eng.)

THESIS COMMITTEE

……………………………………………….. Chairman
(Associate Professor Somchai Prasitjutrakul, Ph.D.)

……………………………………………….. Thesis Advisor
(Associate Professor Wanchai Rivepiboon, Ph.D.)

……………………………………………….. Member
(Assistant Professor Arnon Rungsawang, Ph.D.)

……………………………………………….. Member
(Assistant Professor Pornsiri Muenchaisri, Ph.D.)

……………………………………………….. Member
(Assistant Professor Taratip Suwannasart, Ph.D.)

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

วิวัฒน์ วัฒนาวุฒิ : การสังเคราะห์ข้อกำหนดรูปนัยโดยใช้เครือข่ายอนุภาคความต้องการ. (FORMAL SPECIFICATION SYNTHESIS USING REQUIREMENTS PARTICLE NETWORKS) อ. ที่ปรึกษา : รศ. ดร. วันชัย ริ้วไพบูลย์, 86 หน้า. ISBN 974-17-1532-3.

วิทยานิพนธ์ฉบับนี้ได้นำเสนอแผนการเขียนข้อกำหนดรูปนัยของซอฟต์แวร์โดยใช้กฏการตัดสินใจที่เกิดขึ้นในระบบเป็นตัวกำหนดข้อกำหนดซอฟต์แวร์ ผู้วิจัยได้นำเสนอสัญลักษณ์ทางกราฟิกที่เรียกว่า เครือข่ายอนุภาคความต้องการ (อาร์พีเอ็น) เพื่อใช้ในการอธิบายเงื่อนไขก่อนและการทำงานที่เป็นส่วนหลักและจำเป็นสำหรับระบบซอฟต์แวร์ที่กำหนดขึ้นด้วยกฏการตัดสินใจ เครือข่ายอาร์พีเอ็นประกอบด้วยกลุ่มของอนุภาคและเส้นเชื่อมเพื่อสร้างแบบจำลองเชิงภาพของกฏการตัดสินใจใดๆที่กำหนดขึ้นในช่วงการวิเคราะห์ระบบ งานวิจัยนี้ได้นำเสนอกฏการเปลี่ยนแปลงที่ใช้ในการสังเคราะห์ข้อกำหนดรูปนัย โดยมีโครงข้อกำหนดรูปนัยที่ได้รับการกำหนดขึ้นไว้แล้วและนำมาใช้งานซ้ำได้อีกระหว่างขั้นตอนการสังเคราะห์ข้อกำหนดรูปนัย ข้อกำหนดรูปนัยที่เหมาะสมจะได้รับการสร้างขึ้นอย่างมีระบบเพื่อสนองตอบผู้พัฒนาระบบที่มีความรู้พื้นฐานทางคณิตศาสตร์ไม่มากนัก

ผู้วิจัยได้สาธิตการกำหนดข้อกำหนดรูปนัยที่เขียนด้วยภาษาเซดโดยใช้แผนผังอาร์พีเอ็น การใช้งานของแผนผังอาร์พีเอ็นได้รับการตรวจสอบโดยจัดให้มีการสัมมนาเชิงปฏิบัติการ ผลลัพธ์ที่ได้บ่งชี้ว่าผู้พัฒนาระบบที่มีประสบการณ์ในการเขียนและใช้งานแผนผังการไหลข้อมูลแล้วจะมีขีดความสามารถในการผลิตแผนผังอาร์พีเอ็นได้ครบถ้วนและมีความสอดคล้อง นอกจากนี้ผู้วิจัยได้แสดงกรณีศึกษาที่ใช้งานแผนผังอาร์พีเอ็นสำหรับกำหนดการทำงานแบบคอมโพสิทที่ใช้งานในโปรแกรมประยุกต์ที่มีระบบฐานข้อมูลได้

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

ภาควิชา วิศวกรรมคอมพิวเตอร์	ลายมือชื่อนิสิต...
สาขาวิชา วิศวกรรมคอมพิวเตอร์	ลายมือชื่ออาจารย์ที่ปรึกษา......................................
ปีการศึกษา 2545	ลายมือชื่ออาจารย์ทีปรึกษาร่วม................................

4171822421 : MAJOR COMPUTER ENGINEERING

KEY WORD: FORMAL SPECIFICATION / REQUIREMENTS PARTICLE NETWORK / SPECIFICATION SYNTHESIS / FORMAL METHODS / Z NOTATION

WIWAT VATANAWOOD : FORMAL SPECIFICATION SYNTHESIS USING REQUIREMENTS PARTICLE NETWORKS. THESIS ADVISOR : ASSOC. PROF. WANCHAI RIVEPIBOON, 86 pp. ISBN 974-17-1532-3.

An alternative scheme to formal software specification is explicitly proposed. In our approach, a formal software specification is formally defined as a set of decision rules performed by a software system. We propose a set of graphical notations called Requirements Particle Network (RPN), to describe the essential preconditions and operations needed by a software system according to the decision rules. A RPN consists of a set of particles and edges to construct a visual model of a decision rule during the software analysis phase. In addition, a number of transformation rules are proposed to perform the formal specification synthesis. A set of predefined formal requirements particle definitions is written in prior and reused during the transformation steps. A developer is provided a practical mean to write a formal specification with a brief experience in mathematical background.

In this research, we demonstrate the Z formal specification synthesis using RPN. The usability of the RPN approach is investigated by conducting a workshop. The result indicates that a developer with experience in writing data flow diagram is capable to produce a complete and consistent RPN. Moreover, we show a case study of applying RPN to construct a composite operation to be used in database applications.

Department of Computer Engineering Student's signature...

Field of study of Computer Engineering Advisor's signature...

Academic year 2002 Co-advisor's signature...

ACKNOWLEDGEMENTS

First of all, I would sincerely thank my advisor Associate Professor Dr. Wanchai Rivepiboon for his kind guidance, advice, and encouragement. In particular, I would like to thank my committee - Associate Professor Dr. Somchai Prasitjutrakul, Assistant Professor Dr. Arnon Rungsawang, Assistant Professor Dr. Pornsiri Muenchaisri, and Assistant Professor Dr. Taratip Suwannasart for their reviews and suggestions on this dissertation.

Thanks to the faculties and officers of the Department of Computer Engineering for their supports during my study. Specially thanks to Associate Professor Dr. Prabhas Chongstitvatana, Assistant Professor Nongluk Covavisaruch and Dr. Chaiyong Soonthornphisaj for their encouragement and support. Thanks to my graduate students for helping with the experiments.

Finally, I would not be able to accomplish my study without the loves and supports from my dearest parents and family. Thanks to the special loves and encouragement from my wife and my two lovely sons – Vippy and Winnie.

CONTENTS

Page

ABSTRACT (THAI).. iv

ABSTRACT (ENGLISH)... v

ACKNOWLEDGEMENTS... vi

CONTENTS... vii

LIST OF FIGURES.. x

LIST OF TABLES... xi

1. INTRODUCTION

1.1 Motivations.. 1

1.2 Research Objectives.. 2

1.3 Scope.. 2

1.4 The Structure of this Dissertation................................. 2

2. PRELIMINARIES

2.1 Formal Methods... 4

2.2 Software Specification Methods..................................... 5

2.2.1 Structured Specifications Methods................................ 6

2.2.2 Object-Oriented Specifications Methods........................... 7

2.2.3 Formal Specifications Methods.................................... 7

2.3 Z Notation... 11

2.3.1 Z Schema Structure... 12

2.4 Decision Table... 13

2.5 Formal Specification Synthesis for Database Applications........... 14

2.5.1 Entity Schema Generating Rules................................... 15

2.5.2 Entity Extension Schema Generating Rules......................... 16

2.5.3 Relationship Schema Generating Rules............................. 17

2.5.4 Cascade Insertion Generating Rules............................... 18

2.5.5 Cascade Deletion Generating Rules................................ 20

2.5.6 Cascade Updating Generating Rules................................ 21

Page

3. REQUIREMENTS PARTICLE NETWORK
3.1 Structural Aspects of RPN…………………………………………… 23
3.1.1 An Operation Node in RPN……………………………….. 24
3.1.2 An Edge in RPN…………………………………………… 25
3.1.3 Data Entity Nodes in RPN………………………………… 26
3.2 Behavioral Aspects of RPN……………………………………….. 26
3.3 Formal Definition of Requirements Particle Network…………….. 28

4. FORMAL SPECIFICATION SYNTHESIS USING RPN
4.1 Research Methodology…………………………………………….. 30
4.2 Definition of Formal Specifications……………………………….. 31
4.3 How to Synthesize Formal Specifications…………………………32
4.4 Proof Obligation ……………………………………………………33

5. CASE STUDIES
5.1 Introduction to Video Shop System……………………………….. 35
5.1.1 Decision Table of Video Shop System…………………… 36
5.1.2 Formal Templates of Primitive Operations………………... 38
5.1.3 RPN for Each Decision Rule……………………………… 41
5.1.4 Final Specifications…………………………………………48
5.1.5 Conclusions on Video Shop System……………………… 48
5.2 Introduction to Van Hire System………………………………….. 49
5.2.1 The Entity Relationship Diagram of Van Hire System…….. 50
5.2.2 Formal Specification Synthesis Scheme……………………50
5.2.2.1 Formal Specification of Structural Property……………52
5.2.2.2 Composition of Primitive Operations………………… 52
5.2.2.3 Constructing a New Composite Operations…………… 53
5.2.2.4 Composition Operation Generating Rules…………….. 53
5.2.3 Samples of Composite Operations for Van Hire System….. 54
5.2.3.1 Samples of Decision Table for Van Hire System……. 56
5.2.3.2 Samples of Requirements Particle Definitions……….. 57

Page

5.2.3.3 Samples of RPNs………………………………………… 60

5.2.4 Conclusions on Van Hire System………………………..… 63

6. CONCLUSIONS AND FUTURE WORKS

6.1 Comparison to the Related Works…………………………………64

6.2 Contributions……………………………………………………… 65

6.3 Future Works……………………………………………………… 66

6.3.1 Building the Common RPN Particle Library……………… 66

6.3.2 Applying the RPN for Other Formal Specification Languages…………………………………………………..67

6.3.3 Extending the Time Synchronization Notations……………67

6.3.4 Program Generator from RPN…………………………….. 67

REFERENCES………………………………………………………………… 69

APPENDICES

Appendix A Final Specifications in Case Studies

A.1 The Complete Z Specifications for Video Shop System…………….. 74

A.2 The Sample of Z Specifications for Van Hire System……………….. 79

Appendix B Publications

B.1 International Conferences……………………………………………. 85

B.2 International Journal…………………………………………………. 85

BIOGRAPHY……………………………………………………………….. 86

LIST OF FIGURES

Figure Page

2.1 An example of typical decision table……………………………………13

3.1 A graphical notation of a node of RPN representing
a primitive operation…………………………………………………………24

3.2 A drawing of "message" and "status" edge between two nodes…….. 26

3.3 Sample of the operation nodes and data entity nodes in RPN……… 27

3.4 A Sample of RPN with path P1 and path P2………………………… 28

5.1 Sample of "Store" particle notation………………………………… 38

5.2 Sample of "Search" particle notation……………………………………39

5.3 Sample of "Remove" particle notation…………………………………40

5.4 RPN drawing for rule1 – Insert new video …………………………..42

5.5 RPN drawing for rule2 – Hire a video………………………………..43

5.6 RPN drawing for rule3 – Insert new member……………………….. 45

5.7 RPN drawing for rule4 – Return a video…………………………… 47

5.8 ER diagram of van hire system……………………………………… 50

5.9 Formal specification synthesis scheme for database application……. 52

5.10 A composite operation 'COMPO1'………………………………… 53

5.11 Sample of "Search" particle notation……………………………………57

5.12 Sample of "InsBooking" particle notation…………………………… 58

5.13 Sample of "DelBooking" particle notation………………………….. 59

5.14 RPN drawing for rule1 – Book a van……………………………… 60

5.15 RPN drawing for rule2 – Return a van………………………………62

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

LIST OF TABLES

Table Page

5.1 Decision Table of the Video shop system................................ 36

5.2 Data Dictionary for all Entity Names of Van Hire System 51

5.3 Relationship Dictionary of Van Hire System 51

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER 1

INTRODUCTION

1.1 Motivations

As software projects continue to grow in scale and scope, formal methods are introduced for software development to ensure the accuracy and provable of system. It should be better to have a sound basis in logic and introduction to the logical frameworks used in modeling, specifying and verifying computer systems [1]. Formal methods provide a simple and clear presentation, covering propositional and predicate logic and some specialized logics used for reasoning about the correctness of computer systems. In the early stage of software development process, a formal requirements engineering method [2] is emphasized to elicit the real world needs and several graphical tools will be proposed to accomplish the formal software requirements specification method. In order to exploit formal methods, a lot of formal specification languages are designed to cope with variety of software systems such as concurrent system, real-time system or even the legacy business application.

There is still a wide gap between the current practice of software requirements engineering and the research on formal specification and software formal development. Since a formal specification of software system is difficult to write and understand, a number of active researches are conducted in order to extend the capability of software analyst and designer. Several specific development environment and tools to capture formal specification are proposed, including the relevant specification languages to ease the transformation of formal specification [3], [4] and also the reverse engineering tasks – to transform program codes to formal specifications [5]. We expect that it should be practical for software analyst and designer to be able to investigate their software system specifications in the early stage of the system development. Our work is also motivated by the related research works of Jin [6] and Beeck [7] which the formal specification is generated from requirements definition and then verifiable.

This research aims to provide a practical mean for software analyst and designer to be capable of preparing their formal specifications with brief experiences in mathematical logic background. Our notion is to provide a scheme to obtain a formal specification in the early stage of requirements analysis using graphical tools, called “Requirements Particle Network” (*RPN*), to ease the formalization and its refinements.

1.2 Research Objectives

- To propose an explicit method for formal specification in order to capture user requirements into a mathematical form.
- To propose a set of graphical notations, called RPN, in order to assist requirements engineer or software analyst with a brief experience in formal methods to prepare the formal requirements specification of a software system.
- To propose a set of transformation rules to produce formal specification from the specific RPN’s graphical notations.

1.3 Scope

- The formal specification is synthesized in Z notation.
- Z/EVES is used as our tools and proof system.
- The transformation rules are defined to produce the formal specification.

.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

1.4 The Structure of this Dissertation

This dissertation is organized as follows. Chapter 1 is the introduction of the research. Chapter 2 is the literature surveys on the specification methods, including structured, object-oriented and formal specification methods. The preliminary backgrounds are overviewed as well. Chapter 3 presents our graphical notations called RPN and its formal definitions. Chapter 4 describes our main research topics

on how to synthesize formal specification using RPN. The case studies are summarized in chapter 5 and chapter 6 is our conclusions, contributions, and future works.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER 2

PRELIMINARIES

In this chapter, the related works are surveyed and some preliminary backgrounds are described.

2.1 Formal Methods

With the increasing complexity of software and the greater requirements for reliability, it has become accepted that the old ad hoc informal methods of specifying and implementing software requirements specification are no longer good enough to software development. Even with the use of software engineering and structural design methods, the quality of large-scale software is often still poor.

As an alternative, formal methods provide a mean of specifying computer systems that is unambiguous, concise and well suited to the development of complex software systems for which accuracy and reliability are critical [8]. Using them can help reduce the possibility of errors that occur during the construction of software due to impreciseness and misinterpretation. The mathematical notations are simply used as part of the languages to represent the systems and made them provable. In [9], Jategaonkar mentioned that formal methods technology can provide significant benefits to the software development life cycle. For example, these benefits include more precise requirements, together with early detection of ambiguities and inconsistencies.

Any formal method should have two key components as mentioned in [8], [10], [11] - Formal specification language and its proof system. A formal specification language is designed to have a well-defined firm mathematical semantic to specify the property of a software system. As well, a proof system is generally considered to provide the developer as a deductive apparatus to derive and manipulate any sequence of valid statements of the language from the given axiom of the system so that the specification statements are proved.

Since a heavy mathematical training is needed and it seems difficult to learn, formal methods are practically used in the industry with care guidelines. J. Bowen and M.G. Hinchey proposed ten guidelines in [12] for any software developer who want to gain benefit from formal methods. These guidelines will help ensure that formal methods can be successfully applied in an industrial context. As mentioned in [12], it is important to begin with choosing an appropriate formal notation with the accepted estimated investment cost to use it. The integration of formal methods and the traditional development methods should be considered and gradually conducted with strong quality standards of documentation and procedures.

In [13], Larsen et. al., investigated formal methods into specification and modeling activities of a security critical system's development. The study provided evidence on the effects of introducing formal specification in projects that could benefit from the use of a formal language, such as those that must attain high assurance levels. The study consisted of the parallel development by two separate engineering teams: one team employed conventional development methodology using structural analysis with CASE support tools and the other team employed formal specification language (VDM-SL) wherever they felt it appropriate. The study confirmed the applicability of Bowen's guidelines in [12].

In [14], Hall’s study found the use of formal methods highly effective in detecting more errors at a lower cost per error, than unit testing. In addition to the selection of the appropriate formal specification language, the powerful methods of refinement is needed to cope with the complexity of real systems and the big change in structure that takes place between the system specification and the system architecture.

2.2 Software Specification Methods

In this section, several structured, object-oriented, and formal software specification methods are introduced in brief.

2.2.1 Structured Specification Methods

Over the past two decades, a large number of methods have been proposed for the specification of software systems. The Yourdon approach [18] has been well known for system modeling. Yourdon worked with DeMarco in 1978, Weinberg in 1978, Constantine in 1979, and with Gane and Sarson in 1979 to provide a structured specification method. The context diagram along with a set of dataflow diagrams (DFD) [19] is used to represent external communication and conceptual decomposition of the system respectively. In addition, an entity-relationship diagram (ERD) [19] is used to show the conceptual structure of the data manipulated during the activity.

While, Structured Analysis and Design Technique (SADT) is an another method to functionally decompose the activities to be performed by a system into subactivities [20]. SADT uses activity diagram to represent the external functions, conceptual components (activities), and communication between those components. As well as Yourdon approach, The highest-level activity diagram represents the overall functionality of the system. The lower-level diagrams describe the decomposition into subactivities and their communications. The subactivities are recursively decomposed until a clear understanding of the activities to be performed by the system is reached.

Information Engineering (IE) is a structured method for modeling the information needs of a business proposed by Martin and Finkelstein [18]. The IE method delivers the information systems that a business needs using a function refinement tree to refine the business mission into business functions and refines these into business processes. The ER diagram is used to represent data model. The interesting idea of IE is that the conceptual decomposition of the information system corresponds to the decomposition of the subject domain of the business. The IE method is considered as a data-oriented approach so that business procedures change rapidly, data types are relatively stable [21].

2.2.2 Object-Oriented Specification Methods

Coad and Yourdon [22] use a class diagram to represent the conceptual decomposition of the system into objects. Then, Booch represents the structure of a software system by means of a class diagram and the behavior of the objects by means of state diagram [23]. The sequence diagrams are used to represent object communication. The conceptual decomposition of the software into objects is represented using a class diagram. A dotted cloud symbol is used to represent the classes. The diagram can be annotated by various kinds of constraints. Several diagrams are used in Booch's method. For example, state transition diagram is used to specify object behavior and either collaboration diagram or sequence diagram is used to show the flow of messages in object communication.

The object modeling technique (OMT) was introduced by Loomis et al. [24] and made popular by Rumbaugh et al. [25]. The decomposition of the system into objects is represented by the object model, which is a class diagram. While, the behavior of instances of classes is represented by the dynamic model, which is a statechart variant. Dataflow diagram is used to represent object operations called functional model. The OMT has been modified since 1991 from OMT91 to OMT95. OMT95 can be viewed as a halfway method between OMT91 and the UML.

The Unified Modeling Language (UML) [26] happened as a result of a joint effort by Booch, Jacobson, and Rumbaugh to unify the existing notations for object-oriented software specification. The intention is that it will be used as a diagram convention in object-oriented software specification and develop into the standard for practitioners.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

2.2.3 Formal Specification Methods

As the structured and object-oriented specification methods are being exploited conventionally, an alternative of formal methods in specifying is accepted. Formal specification methods are being used to improve the quality of written specification, and particularly to eliminate errors at an early stage of software development [27].

The objective of software engineering is to model software systems to support effective system synthesis and analysis. The traditional approach is using semi-formal models and accompanying design methodologies. Until now, formal models represent an alternative to semi-formal models supporting precise representation and proof capabilities. Semantics are simply a mapping from objects in the problem domain to well form formula in the formal language [15]. We can model any known characteristics using the semantic mapping mentioned. Thus, this particular formal system becomes a formal model of target problem domain.

In general, formal specification [16] is a set of valid statements written in mathematical notations to describe properties in which an information system must have, without constraining the way in which these properties are achieved. The mathematical statements in mathematical logic [17] describe what the system must do without saying how it is to be done. The developer of a computer system may not have to deal with the mass of detailed program code, or to speculate about the meaning of phrases in an imprecisely written description.

With a given formal specification of a software system (written in any formal specification language such as Z, CSP, LOTUS, etc.), the states of the software system operation are formally described and the refinement of system specification can be conducted to fit the requirements or needs.

To compromise between conventional methods and formal methods, Zhang [28] experimented the combination structured specification method using DFDs and formal specification method using Interval Temporal Logic (ITL) for requirements analysis. He conducted the structured analysis with DFDs methods, which provided the readability of the overall system definitions. Then, ITL has been used to cope with the specification, design, verification and validation of real-time system in his case study.

Formal specification methods are being used in specifying important software systems. In general, these specifications are expressed as a set of operations to be implemented, which includes the types of values to be input to and output from each operation. Jones [29] termed two distinct types of methods as model-oriented and property-oriented. In the model-oriented approach the specification includes an abstract model of the internal state of the component being specified, so that the behavior of individual operations is defined in terms of their effect upon the

components of the state. In the property-oriented approach (commonly called the algebraic approach) the behavior of individual operations is defined by stating the equivalent equations between different combinations of operations, with no explicit model of the state being necessary.

The Vienna Development Method (VDM) is a formal software specification method. VDM provides three components: a notation for expressing software specification; an inference system for constructing proofs of correctness; and a methodological framework for developing software from a specification in a formally verifiable manner [8]. In VDM, the high-level abstract data types of the original specification can be moved to the data types of the target programming language, which is called "reification". As well, the operational decomposition of specified functions and operations can be converted into more implementable versions for the target language. At present, VDM does not include the ability to specify concurrent processes and this seems to be a disadvantage in the specification of communication protocol and other areas where such constructs would be useful.

The Z specification method uses Z notation along with Z Theorem Prover. The Z notation is based on typed set theory and first-order logic. Z provides a construct, called a schema, to describe a specification's state space and operations. A schema groups variable declarations with a list of predicates that constrain the possible values of a variable. Several formal methods tools are available for Z. For example, Z/EVES [30] supports the analysis of Z specifications in several ways: syntax and type checking, schema expansion, precondition calculation, domain checking, and general theorem proving.

The B-Method is another formal method by Jean-Raymond Abrial [31],[32] uses the paradigm of Abstract Machine (AM) as unit of specification. An AM consists of summarily in two main parts. The first part is the static state of the system, which is described using the variables and the invariant of the variables. The second part is the dynamic behavior of the system that is expressed through its operations using the paradigm of Abrial's generalised substitution language (GSL). The B-Method has a particularly strong notion of layered development [33], which allows a complex development to be decomposed in a rich variety of ways using a small number of basic constructs. This makes the refinement of industrial-scale systems practical.

DisCo (Distributed Co-operation) [4],[34] is a formal specification method for reactive systems. It incorporates a specification language, a methodology for developing specifications using the language, and tool support for the methodology. Currently the support tools include an animation facility for executing specifications, a tool for visualizing execution histories as scenarios, and a link to a mechanical theorem prover for verification. The method has a solid formal basis, but the specification language uses concepts and notations familiar to people with a conventional software engineering background. DisCo is an incremental process. Disco starts with very simple behavior, and gradually adds details until the specification is at the desired level. Development focuses on collective behavior, that is, how objects co-operate. Developing a DisCo specification can be compared to carving a shape out of a block of wood. The original block of wood represents all the possible behaviors, and removing a piece of wood corresponds to a design step that disallows some of the behaviors. In early stages of the process we get a rough sketch, in which details are then gradually added. DisCo employs the closed world principle, meaning that a DisCo specification always describes a system together with its environment.

The CafeOBJ Object-Oriented Methodology for component-based specification and verification was proposed in [35]. The methodology exploited CafeOBJ [36] behavioral abstraction paradigm to define the component-based specification in terms of new algebraic logic based. The composition of objects or components of the system are represented using UML's class diagrams as a static view of the system. The compound objects expected to be decomposed into a set of non-compound objects (objects with no components) called base level objects. The ADJ diagrams are used to represent the dynamic connection of the objects. CafeOBJ provides a verification process through its proof tree using built-in rewrite engine. A network based environment of CafeOBJ development called CAFE [37] was proposed for supporting systematic creation, checking, verification, and maintenance of formal specifications in the industrial world.

2.3 Z Notation

Z is a formal specification language. The Z notation is used to describe the behavior of a system. It uses the simple mathematics, consisting of first order predicate logic and set theory. However, it offers a very elegant way of structuring the specifications to be manageable modules called schema. Z grew out of work at Oxford University's Programming Research Group and it has been applied to numerous projects involving both software and hardware in a number of application areas, such as financial, security, safety, critical, etc.[54],[55]. One of the disadvantages of Z is that it provides little in the way of methodology. The word methodology mentioned means two things [56]: a management methodology describing how and to what Z can be applied, and how to control the process of developing a system from a Z specification. Second, a formal methodology which describes a set of rules for transforming a Z specification into another more detailed description.

The Z specifications can be written in the "states-and-operations" style. In this style a system is described by specifying operations which describe changes to the state of the software system. The state of the system and the operations are written in Z using schemas which structure the specification into convenient components. A calculus is provided to combine the schemas in appropriate ways, and this schema calculus helps to structure the specifications.

Z uses propositional and predicate logic to express relationships between the components of a system. The propositional logic used contains a number of connectives, such as negation, conjunction, disjunction, implication, and equivalence. The predicate logic introduces the quantification into the language, together with free and bound variable, such as universal quantification (for all) and existential quantification (there exists).The set theory is also used to define as relations, functions and sequence. Membership, the empty set, equality, and subset are all defined. The power set constructor, Cartesian products, union, intersection, and difference are all available with the language, as well.

Z provides a built-in type, namely the type of integers Z. One way to build further types is to simply declare them. A given set is a declaration of the form

[*STUDENT*]

which introduces a new type *STUDENT*. It defines a set of *STUDENT*, but at this stage no information is given about its values and the relationships between them. Types can also be constructed from existing types in a variety of ways. These include Cartesian products, for example, $STUDENT \times \mathbb{Z}$ is a type consisting of ordered pairs. Thus, if *Surat* is of type *STUDENT* then $(Surat,\ 1) \in STUDENT \times \mathbb{Z}$. The power set constructor $\mathbb{P}$ is used to define the type of collections, for example, $\mathbb{P}\ STUDENT$ is the set of student members likes *{Surat, Supa}*. Another important type constructor is the free type. It is defined as a given type with the additional constraints and is normally declared in a BNF-like notation, for example, *GENDER ::= male | female* denotes a type *GENDER* containing exactly two different constants *male*, and *female*.

2.3.1 Z Schema Structure

Z schema is the best known for. Schema consists of two main parts: declaration part and axiom part. Declarations may be split across lines or put on the same line, separated by semicolons. A Z schema can be written in either box format or horizontal format.

Here is the horizontal format of a state schema named *Van*

Van ≙ *[RegistrationNo : STRING;*
Model : STRING;
Class : CLASSNAME]

Here is the box format of *Van*

—*Van*————————
RegistrationNo : STRING
Model : STRING
Class : CLASSNAME
——————————

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

A schema can be included in another schema in order for a new schema to be reused the existing schema. In addition, a new schema can be constructed with the combination of the existing schemas, for example, using schema conjunction, disjunction, and composition, etc.

There are a number of tools to support Z, such as *OZ, CaDZ*, *Object-Z.* They offer varying degrees of functionality, such as syntax and type checking, theorem proving, and animation. In our research, *Z/EVES* is used as the tool to check the correctness of types and syntax.

2.4 Decision Table

A decision table [38] is a tabular form for specifying decision logic. A decision table provides an effective means of defining both the problems and their corresponding logical solutions.

The typical example shown in figure 2.1 is a decision table. Each vertical combination of preconditions and actions is called a decision rule. The advantages of using decision tables become even more apparent in larger, more complex situations.

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	..	Rule R
Precondition 1	X	X					X
Precondition 2		X	X				
Precondition 3	X		X	X			X
...							
Precondition P					X		X
Action 1	X						
Action 2		X	X				X
Action 3	X	X			X		
...							
Action A			X		X		

Figure 2.1: An example of typical decision table.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

A decision table can be divided into four quadrants. The upper left quadrant, called the condition stub, should contain all those conditions being examined for a particular problem segment. The condition entry is the upper right quadrant. These two sections describe the set, or string, of conditions that is to be tested. The lower left quadrant, called the action stub, contains a simple narrative format for all possible actions resulting from the conditions listed above the horizontal line. Action entries are given in the lower right quadrant. Appropriate actions resulting from the various combinations of responses to the conditions will be indicated in the action entry.

A decision table displays any precondition that must be satisfied before any prescribed actions will be performed. They are becoming very popular in computer programming and system design as a tool for organizing logic, especially when attempting to handle very complex situations, and to be able to account for every possible combination of conditions. Furthermore, the modification of the requirements of a software system is easily documented by the unique form of the problem statement in decision tables. Decision table is easy to learn because of its simple structure and efficiency in using it can be reached with little experience.

2.5 Formal Specification Synthesis for Database Applications

This research was our previous work and reported in [39]. We proposed the rules of generating formal specifications of the structural property of a relational data model. The entity relationship diagram and its data dictionary, commonly prepared during the early stage of software development, are used as original data model definitions.

An entity relationship diagram is transformed into three categories of schemas in Z: *Entity schema, Entity Extension schema*, and *Relationship schema*. The Entity schema represents all of the attributes and types in an entity. In the meantime, the Entity Extension schema defines the constraints of structural property such as primary key, foreign key, as well as the integrity constraints. The Relationship schema defines the relations between entities along with the cardinality ratio - one-to-one, one-to-many, and many-to-one relationship. The transformation rules are briefly presented in the following subsections.

2.5.1 Entity Schema Generating Rules

Each entity *ENT* in an entity relationship diagram can be translated into an entity schema *EN* in Z with the following conventions:

1) An entity schema *EN* is named by using the original entity name *ENT*.
2) All of the attributes *AN* shown in ENT_{ATT} are listed as the variables in declaration part of the entity schema *EN*.
3) All of the attribute types *AT* shown in ENT_{ATT} are defined as given sets (abstract data type) in Z.
4) All of the constraints on the attributes *CS* shown in ENT_{ATT} are listed as the invariant predicates in axiom part of the entity schema *EN*.

The formal template of the entity schema is shown as follows:

```
[AT1, AT2, ..., ATn]
┌─EN──────────────────
│ AN1 : AT1
│ ....
│ ANn : ATn
├─────────────────────
│ CS
└─────────────────────
```

For example, the entity VAN has four attributes – registrationNo, model, vanclassFK, and usedPeriod. vanclassFK is the foreign key from entity VANCLASS. The constraints on attribute usedPeriod is "usedPeriod ≤ 10". By using entity schema generating rules, the entity schema for VAN is generated as follows:

```
[ STRING, CLASSNAME ]
┌─VAN─────────────────
│ registrationNo : STRING
│ model : STRING
│ vanclassFK : CLASSNAME
│ usedPeriod : Z
├─────────────────────
│ usedPeriod ≤ 10
└─────────────────────
```

2.5.2 Entity Extension Schema Generating Rules

An entity extension schema *ENExt* is generated for each entity *ENT* in an entity relationship diagram in order to describe the structural property constraints. *ENExt* is generated in Z with the following conventions:

1) An entity extension schema *ENExt* is named by the original entity name *ENT* with postfix "*Ext*".
2) Define a variable, named after the entity schema *EN*, as finite set of *EN*.
3) Define primary key by using definition of primary key mentioned earlier in the axiom part of the entity extension schema *ENExt*.
4) If there exist the foreign keys ***child*** from parent entities, *ParentEnt,* then define each foreign key $child_i$ by using definition of foreign key mentioned earlier in the axiom part of the entity extension schema *ENExt*.

The formal template of the entity extension schema is shown as follows:

```
┌─ENExt──────────────────────────────────
│ en : F EN
├────────────────────────────────────────
│ ∀e, e2 : en | e1 ≠ e2 • e1.key ≠ e2.key
│ ∀child1 : en • ∃parent1 : ParentEnt1 •
│         child1.foreignkey = parent1.key
│ ∀child2 : en • ∃parent2 : ParentEnt2 •
│         child2.foreignkey = parent2.key
└────────────────────────────────────────
```

For example, the entity extension schema VANExt is generated for the entity schema VAN. The primary key of the entity VAN is explicitly defined and shown in the first predicate of VANExt. The attribute named registrationNo is used as the unique primary key. The attribute name className in VAN is the foreign key from VANCLASS.

$$
\begin{array}{|l}
\textit{VANExt} \\
Van : \mathbb{F}VAN \\
\hline
\forall v_1, v_2 : Van \mid v_1 \neq v_2 \bullet \\
\quad v_1.registrationNo \neq v_2.registrationNo \\
\forall child : Van \bullet \exists parent : Vanclass \bullet \\
\quad child.vanclassFK = parent.className \\
\hline
\end{array}
$$

2.5.3 Relationship Schema Generating Rules.

A relationship schema *RelationshipR* is generated for each relation in an entity relationship diagram using the following conventions:

1) A relationship schema *RelationshipR* is named by the original relation named *REL* with prefix "*Relationship*".
2) Define the cardinality ratio of the original relation *REL* according to relationship dictionary. The ratio may be one-to-many, many-to-one, or one-to-one relationship. The definitions of cardinality ratio of relationship mentioned in section 2 are used.
3) All of the constraints on the relation *CS* are listed as the invariant predicates in axiom part of the relationship schema.

The formal template of the relationship schema for one-to-many relationship is shown as follows:

$$
\begin{array}{|l}
\textit{RelationshipR} \\
R : \mathbb{F}(EN_1 \times EN_2) \\
\hline
\forall r_1, r_2 : R \bullet second(r_1) = second(r_2) \Rightarrow \\
\qquad\qquad first(r_1) = first(r_2) \\
CS \\
\hline
\end{array}
$$

For example, the relation MAKES is one-to-many relationship between CUSTOMER and BOOKING. The definition of one-to-many relationship mentioned earlier is used. Since binary relation is focused, we implement a relation as an ordered

pair and use Z notation - first(), second( ) to implement domain and range respectively.

RelationshipMakes
makes : $\mathbb{F}$ *(CUSTOMER X BOOKING)*

$\forall m_1, m_2 : makes \bullet second(m_1) = second(m_2) \Rightarrow first(m_1) = first(m_2)$

The next sections define the rules of generating formal specifications of the behavioral property of the relational data model. We intend to exhaustively generate all of the primitive operations for each data entity in the entity relationship diagram. The rules provide cascade insertion, cascade deletion, and cascade updating operations. The referential integrity, which is the mandatory constraints of relational data model, is maintained with these cascade operations.

2.5.4 Cascade Insertion Generating Rules

An insertion operation schema in Z is generated for each entity *ENT*. When a new instance of *ENT* is inserted, the structural constraint properties defined in entity extension schema are preserved such as the uniqueness of the primary key, the foreign key, etc. Moreover, the insertion operation schema can do cascade insertion, which will maintain the referential integrity during the operation.

The schema is generated as follows:

1) The insertion operation schema is named by using the entity name *ENT* with prefix "*Insert*".
2) Include the entity extension schema into the insertion schema, *ΔENExt*.
3) Define a variable for new value of the instance.
4) If there exists the parent entity of *ENT* then include insertion operation schema of the parent entity *InsertP* in the declaration part. The parent entity of *ENT* is recognized by the appearance of foreign keys in entity

ENT. Each foreign key indicates that *ENT* is the child entity of the original entity of the foreign key.

5) Define the next state of the entity *ENT* using union operation in Z.

The formal template of the insertion operation schema is shown as follows:

```
┌─InsertEN──────────────────────
│ ΔENExt
│ newEN? : EN
│ InsertP
├──────────
│ en′ = en ∪ {newEN?}
└───────────────────────────────
```

For example, the insertion operation schema InsertVAN is generated by the cascade insertion generating rule. The entity extension schema VANExt is included to provide the constraint properties of VAN and a variable named newVAN? is defined for the new value of the new van. We recognize that VAN has a foreign key called 'VanclassFK' so that the original entity of the key, that is VANCLASS, is needed for the cascade insert. Thus, the insertion operation schema InsertVANCLASS is invoked as the prerequisite operation of InsertVAN as shown below.

```
┌─InsertVANCLASS────────────────
│ ΔVANCLASSExt
│ newVANCLASS? : VANCLASS
├──────────
│ Vanclass′ = Vanclass ∪ {newVANCLASS?}
└───────────────────────────────
```

```
┌─InsertVAN─────────────────────
│ ΔVANExt
│ newVAN? : VAN
│ InsertVANCLASS
├──────────
│ Van′ = Van ∪ {newVAN?}
└───────────────────────────────
```

2.5.5 Cascade Deletion Generating Rules

A deletion operation schema in Z is automatically generated for each entity *ENT* as well as insertion mentioned earlier.

The schema is generated as follows:

1) The deletion operation schema is named by using the entity name *ENT* with prefix "*Del*".
2) Include the entity extension schema into the deletion schema, $\Delta ENExt$
3) Define a variable to hold the value of the instance to be deleted.
4) If there exists the child entity of *ENT* then include deletion operation schema *DelC* of the child entity in the declaration part.
5) Assign the target value to be deleted in the child entity. All of the instances of child entity, which carry the same target value in the foreign key will be deleted.
6) Define the next state of the entity *ENT* using set difference operation in Z.
7) An extra operation schema called *DelC* is generated in case of the cascade deletion. As mentioned in rule 5, all of the instances of child entity that carries the target value in the foreign key are deleted.

The formal template of the deletion operation schema is shown as follows:

DelC

$dC? : C$
$\Delta CExt$

$c' = c \setminus \{dC?\}$

DelEN

$\Delta ENExt$
$dEN? : EN$
$DelC$

$dC?.FK = dEN?.PK$
$en' = en \setminus \{dEN?\}$

For example, an instance of VANCLASS with className = "4-Wheel Drive Class" is to be deleted. The target value will be passed into DelVAN schema via dVAN?. All of the instances of VAN, which carry the value "4-Wheel Drive Class" in attribute name vanclassFK will be deleted. No more "4-Wheel Drive Class" is available for van hire system. We can recognize the child entity by tracing all of the one-to-many relationship from VANCLASS to all of the related entities. The sample of DelVANCLASS is generated as follows:

```
—DelVAN——————————————————
dVAN? : VAN
ΔVANExt
—————————————————————————
Van′= Van \ {dVAN?}
—————————————————————————

—DelVANCLASS—————————————
ΔVANCLASSExt
dVANCLASS? : VANCLASS
DelVAN
—————————————————————————
dVAN?.vanclassFK = dVANCLASS?.className
VanClass′= VanClass \ {dVANCLASS?}
—————————————————————————
```

2.5.6 Cascade Updating Generating Rules

An updating operation schema is generated for each entity *ENT* using the deletion and insertion operation schema. We consider that the update operation is the sequences of deleting the unwanted instances and inserting the new values. The integrity properties are preserved and the cascade update is provided as follows:

1) The update operation schema is named by the entity named *ENT* with prefix "*Update*".
2) Invoke the deletion operation and insertion operation using the sequential schema composition

UpdateEN

DelEN; InsertEN

For example, the model of the van which has registrationNo = "CA2034" is to be corrected. The instance of VAN is deleted and the new value is inserted. The sequential composition of the schema (';') is used.

The sample of *UpdateVAN* is generated as follows:

UpdateVAN

DelVAN; InsertVAN

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER 3

REQUIREMENTS PARTICLE NETWORK

In this chapter, the *Requirements Particle Network (RPN)* is clearly described. We intend to introduce some basic concepts of our proposed network in the narrative way and conclude with the formal definitions at the end of this chapter. In section 3.1, we describe the structural building blocks and the connections of a RPN. Then, the behavioral part of RPN is described in section 3.2. Moreover, the definition of RPN has been worked out and finally defined in section 3.3.

3.1 Structural Aspects of RPN

A RPN is written and considered as a hypergraph, which a set of nodes and connecting edges are shown. In practical, a RPN represents a requirements primitive - a small event occurred in a software system. We will discuss on how to specify the requirements primitives of a software system in the later chapter. The basic building blocks of a RPN are the nodes appeared and connected with the edges. We provide two kinds of nodes: an operation node and a data entity node. An operation node denotes a primitive operation needed to be exploited in the particular event or the requirements primitives. A primitive operation is an atomic which is the smallest part of requirements primitive.

For example, the atomic operations that perform store-and-retrieve activities in a software system can be *insert*, *delete* and *update* operation in order to manipulate the state of the target software system. While, a data entity node denotes a data value or data set to be used as a data source and a data sink in a RPN. A data source is usually depicted as an origin of the input data, and vice versa, a data sink is considered as a destination of the output data. Each node is connected to an adjacent node with an edge to gradually depict the structural skeleton of a RPN.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

3.1.1 An Operation Node in RPN

An operation node, in RPN, has ports for input, output, and condition input and output. Every port is associated with a type, which specifies the set of data values that the port can hold. We show a graphical notation of an operation node of RPN in figure 3.1.

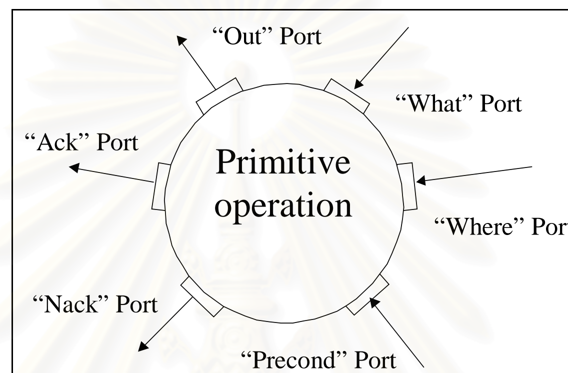


Figure 3.1. A graphical notation of a node of RPN representing a primitive operation

We introduce a set of ports for input called "What", "Where", and "Precond" ports and a set of ports for output called "Out", "Ack", and "Nack" ports.

- A "What" port is usually used to specify the input data value to be performed by the primitive operation. For example, a "Retrieve" operation will perform the seeking of the target data value specified by "What" port.
- A "Where" port is used to specify the target data set or data container to be involved in the primitive operation. For example, a "Retrieve" operation will perform the seeking of a data value specified by "What" port from the target data set or data container named by "Where" port.
- A "Precond" port is used to hold the invariant of the primitive operation. In practical, the invariant of the primitive operation will be asserted before the operation occurred, so we called this invariant as "precondition". The precondition should be asserted and its result must be true.

- An “Out” port is used to forward the message from the operation. The message may be a data value or data set of any type.
- An “Ack” port is the successful consequence of the operation within a node. When an operation is performed and considered as a success task, the data value of true is forwarded, otherwise, the data value of false is forwarded instead.
- A “Nack” port returns the opposite value of an “Ack” port. We may use the data value from “Nack” port to indicate the failure of the operation when it holds the value of true.

3.1.2 An Edge in RPN

The only way to communicate with the outside world of a RPN node is to send and receive any message via ports. The input messages will be used and the output messages are produced. To complete a RPN, the ports of two nodes are connected by edges. A RPN has two different kinds of edges, of which, we address “message” edge and “status” edge.

- A “message” edge is used to specify the direction of communication of data values from a source operation node to a sink operation node. Each message edge has exactly one port at its head and exactly one port at its tail.
- A “status” edge is specifically used to specify the direction of communication of Boolean value (true or false) or a predicate from a source port to a sink port.

In figure 3.2, we show the drawing of “message” edge and “status” edge between two nodes.

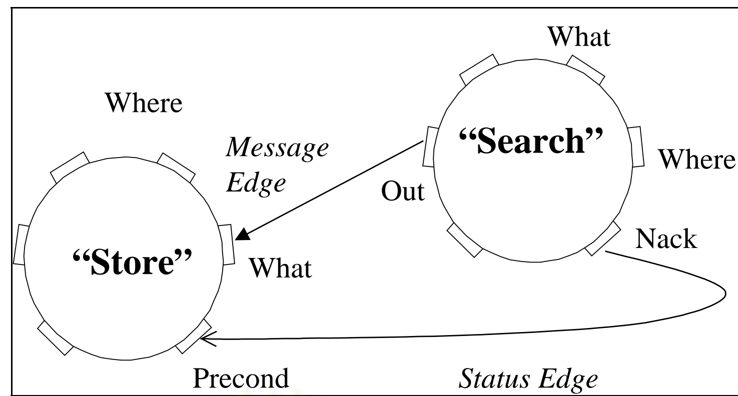


Figure 3.2 A drawing of "message" and "status" edge between two nodes

3.1.3 Data Entity Nodes in RPN

In practice, we can draw any terminal node (a node which either no input or no output) as a data entity node – a data source to be used to feed the data value to the next node or the data sink as the destination of the output from any node. For example, a data entity node called "Video Title" is the data source of operation node called "Retrieve". The "Retrieve" node expects a data value from "Video Title" to be able to perform the search and help us to know whether the expected video title has already been in the video stock. From the figure 3.3, we assign a data entity node called "Video Stock" to be our target container when the operation begins to search.

We simply draw a message edge between "Video Title" node and "Retrieve" node via "What" port. In similar, we have a message edge between "Video Stock" node and "Retrieve" node via "Where" port either.

3.2 Behavioral Aspects of RPN

In figure 3.4, we show a sample of RPN which performs a registration of a new video title into the current video stock. The following narrative items introduce the components of the RPN:

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

- "Retrieve" node is an operation node that performs the searching of an input video title from the target video stock. It forwards the video title to the next node via "Out" port. The "Act" port forwards a Boolean value as "true" when the video title is founded. Otherwise, the "Act" port will return "false". The "Nack" port forwards the opposite Boolean value of "Ack" port. It means that the "Nack" port will be true when the video title is not found.
- "Store" node is another operation node that performs the inserting of an input video title into the target video stock. The "Precond" port of "Store" will be asserted before the inserting. When the inserting is successfully done, the "Ack" port will return "true." Otherwise, the "Nack" port will be true.
- "OutDevice" node is an operation node that display the message specified by the "What" port on any device specified by the "Where" port. Usually, we can use the node to forward any desired message to a monitor CRT.

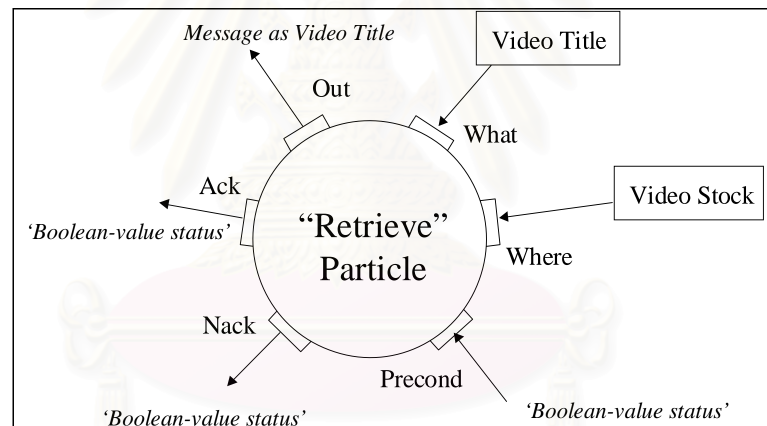


Figure 3.3 Sample of the operation nodes and data entity nodes in RPN

The behavior of a RPN is determined by the behavior of individual nodes and their connectivity, which determines the flow of messages and status values in Boolean. In figure 3.4, there are two possible paths: P1 and P2. In this case, both P1 and P2 are used to describe the behavior of the RPN. They specify how the RPN react to the outside world. The mentioned RPN will try to retrieve a video title from specified video stock. It will display a message "Existing Title" on a monitor CRT, if there exists the video title in video stock. Otherwise, it stores the video title into video stock and finally displays message "Register Done" on a monitor CRT. We would

like to make it more clearly that a data entity node may be considered as an abstract data sink, in this case – a monitor CRT.

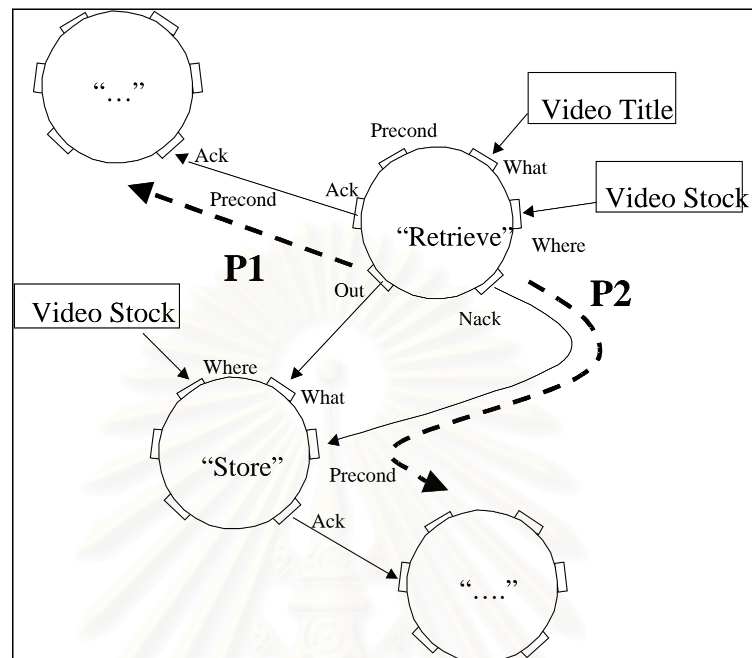


Figure 3.4 A sample of RPN with path P1 and path P2

3.3 Formal Definition of Requirements Particle Network

A *Requirements Particle Network* [52] is a well-defined graphical diagram that provides the composition rules to compose a set of primitive operations into a complex operation. We consider each primitive operation on data entity such as insertion, deletion, etc. as a particle in the network. We briefly overview the formal definitions of the requirements particle network in this section.

Definition 1: (Data Entity)

A data entity *DE* is the representative of data element or a set of data elements. Data entity *DE* is defined as an ordered pair <*Name, Type*>. Let *de* be a data entity, *de.Name* refers to attribute "*Name*" and *de.Type* refers to attribute "*Type*". ■

Definition 2: (Requirements Particle)

A requirements particle *RP* is an atomic node that performs a specific task. A requirements particle is formally defined as a collection of ordered *attributes <Name, What, Where, Precond, Out, Ack, Nack>*. Let *rp* be a requirements particle, p.*Name* refers to attribute "*Name*" while the rest of attributes are referred as *rp.What*, *rp.Where*, *rp.Precond*, *rp.Out*, *rp.Ack*, *rp.Nack*, respectively. ■

The last six attributes are called communication ports. Each requirements particle communicates to outside world via these communication ports. A number of messages are received via "*What*" and "*Where*" port while precondition status is obtained via "*Precond*" port. "*Out*" port is used by each particle to forward postconditions to the successive particles. Boolean-value status will be transmitted to successive particles as well via "*Ack*" and "*Nack*" port.

Definition 3: (Requirements Particle Network)

A requirements particle network is a tuple *RPN = (V, P, D, ES, EM)*. We define $V = P \cup D$. *P* is a set of nodes called requirements particle nodes. *D* is a set of nodes called data entity nodes. *ES* is a set of edges on *P X P*, called status edges. *EM* is a set of edges on *V X V*, called message edges. Let $p,q \in P$ and $d \in D$, edges are written as $p \xrightarrow{Status} q \in ES$, $p \xrightarrow{Message} q \in EM$, $d \xrightarrow{Message} q \in EM$. A requirements particle network is a network of requirements particles and associated data entity. Practically, software functional requirements specification can be represented by a set of requirements particle networks. ■

Definition 4: (Final Formal Specification from RPN)

The final formal specification is generated from *RPN* as a complex formal specification *CFS = (FS(RPN), CRP(RPN))*, where *FS(RPN)* represents a set of specification modules defined on each *RPN* and *CRP(RPN)* is a connection description for *FS(RPN).* A set of formal specification modules *FS(RPN)* defined as $\{FST(p,RPN) \mid p \in P\}$, where *FST(p,RPN)* is a transform function defined by transformation rule 1 (defined in the next chapter). A well-defined requirements particle definition of formal specification is assigned to match each p.*Name*. *CRP (RPN)* can be obtained by transformation rule 2 (defined in the next chapter). ■

CHAPTER 4

FORMAL SPECIFICATION SYNTHESIS USING RPN

In this chapter, we present our main research approach to synthesize formal specification using the new graphical notations, called "Requirements Particle Network" or RPN in short. In section 4.1, we begin the overview of the research methodology to indicate the main steps on the specification techniques. The definitions of our related components are described in section 4.2 and how to synthesize formal specification is described in section 4.3. The proof obligation guided by Z/EVES [53], is briefly described in section 4.4.

4.1 Research Methodology

This section introduces our research methodology. One of our research objectives is to propose a scheme for preparing formal specification during the early stage of the software development lifecycle. The users' needs are assumed to be elicited using some relevant requirements elicitation techniques and documented into a form of textual, formatted description, so-called "Informal requirements documents." In requirements engineering research areas, several approaches are proposed. Among those techniques, scenario-based approach is one of the most famous [40], [41]. Generally, all of the flows of actions in a software system expected to be completely documented.

With the conventional way of semi-formal modeling techniques, the informal requirements will be converted into diagrams, charts and some formatted descriptions. For example, a particular software system may be described with a set of data flow diagrams, entity relationship diagram, and their data dictionaries [19] or a set of Use Case diagrams, and their collaborations [26], [19].

At this stage, the software analyst may try to write the formal specifications of the user requirements to model the software system. However, as we mentioned

earlier, the formal specification of a software system is still difficult to be prepared from the mentioned diagrams and charts.

In our research, we propose an alternative scheme to prepare the formal specifications in the following steps:

- Rewrite the textual and formatted descriptions of the user requirements into a decision table [38].
- Define the primitive operation needed.
- Define the requirements particle definition for each primitive operation defined in the previous step.
- Draw a RPN for each decision rule in the decision table.
- Apply the transformation rules to convert RPN into a set of Z specifications.
- Combine the final formal specifications of all decision rules.

We assume that the textual and formatted descriptions of the user requirements have already been completely elicited. The descriptions of user requirements are rewritten into a decision table – a set of conditions and actions. A set of primitive operations is defined to perform the actions needed for each decision rule in our decision table. A requirements particle definition – a substitutable Z specification, are defined for each primitive operation.

The requirements particle networks of a particular software system can be simply drawn from the corresponding decision rules. A final formal specification of software system will be expected as a result using the transformation rules.

4.2 Definition of Formal Specifications

Definition 5: A specification of software functional requirements *SPEC* is considered as a collection of rules that the target software system is obliged to follow as to accomplish user's needs. To provide a formal framework of how to write a unambiguous functional specification for software requirements engineer, the software functional requirements specification is formally defined as *SPEC* =

({RPN}, CPS) where *{RPN}* is a set of requirements particle networks, and *CPS* is a connection description for *{RPN}*. *RPN* and *CPS* are defined in the next paragraph. In our approach, a connection description is defined as a schema in Z. ■

4.3 How to Synthesize Formal Specifications from RPN

In order to synthesize a formal requirements specification, the step-by-step procedure is defined as follows:

- Develop SPEC using decision table approach.
- Draw requirements particle network for each rule in SPEC.
- Transform each RPN into formal specification of each primitive with a connection description, using transformation rule 1 and 2.
- Compose the final formal specification, using transformation rule 3.

Transformation Rule 1: Transform Function *FST(p,RPN)*

This rule selects the formal definition which represents each requirements particle *RP* and substitute all of the state variables within the specification with the name of data entity and the outcome from the previous *RP*, as follows:

- Select requirements particle definition that matches p.*Name.*
- Substitute all the state variables named "*What*" with d.*Name* where d $\xrightarrow{Message}$ p.*What*.
- Substitute all the state variables named "*Where*" with d.*Name* where d $\xrightarrow{Message}$ p.*Where*.
- Substitute the state variables named "*Precond*" with the predicates of status edge from q.*Ack* or q.*Nack* where q ∈ P and q $\xrightarrow{Status}$ p.*Precond.*

Transformation Rule 2. Generating *CRP(RPN)*

The rule generates the predicates that glue all of the available partial specification of each *FST* together, as follows:

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

- Find p where p,q $\in P$ and there exists no such q that q $\xrightarrow{Status}$ p.*Precond*.
- Define *CRP(RPN)* as $FST(p,RPN) \wedge (CRP(RPN_{Ack}) \vee CRP(RPN_{Nack}))$ where RPN_{Ack} is the sub network connecting to p.*Ack* and RPN_{Nack} is the sub network connecting to p.*Nack*.

Transformation Rule 3. Generating *CPS*

Let rp be a requirements primitive, *CPS* is defined as $\bigvee_{rp \in RP} rp$

4.4 Proof Obligations

We use Z/EVES as the prover tool and conduct three categories of proofs: Syntax and Type checking, Goal proof, and Test Case checking, recommended in [53]. All of the schemas in the final formal specification are converted into Z-Latex format (Z/EVES input format) for the proof processing. Z/EVES can be used incrementally: as each paragraph of a specification is entered, it can be immediately checked and syntax and type error messages are reported.

Since the Z style of specifying operations is relational, an operation is specified as a relation between initial and final states. This style leaves the precondition of an operation implicit that may cause the unsatisfied result. For example, an operation *OP1* as follows:

OP1
$n, n' : \mathbb{N}$
$n' = n-1$

OP1 describes only situation where the initial value of n is non-zero. The conventional interpretation of this is that if *OP1* is invoked with n = 0, there are no guarantees about what might happen. The operation might fail when $n' \notin \mathbb{N}$. Such an operation may result in a state that does not satisfy the predicate part of the state schema.

Z/EVES provides an alternative for the Goal proof using precondition reference, we set up the expected goal predicate to be proved as the following theme:

For each operation OP:

OP | *[ΔEN; in?:IN; out!:OUT]*

The goal predicate is defined as:

A EN; in?:IN; • pre OP

The Z/EVES proof steps transform the goal predicate to an equivalent predicate. The implicit precondition in Z of each OP will be explored before proof process. The additional explicit predicates may be needed during the proof in order to achieve the goal predicate.

For the Test Case checking, we have to specify more details on how to represent the given type set such as STRING, FUELTYPE, etc. We define the STRING as a sequence of characters while a character is defined as an integer value of ASCII code. A set of test cases is prepared to test all of the operations generated. We then observe the result of the test case from the status provided by Z/EVES. The proofs show the correct results so that all of the proposed rules are practical to be used.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER 5

CASE STUDIES

In this chapter, we present the case studies to illustrate formal specification synthesis using RPN. In section 5.1, we gradually describe how to synthesize formal specification in Z with the first case study called "Video Shop System", using the definitions and transformation rules defined in previous chapter. In section 5.2, we intend to experiment on how to apply RPN to generate the composite operations on realistic database definitions in the second case study called "Van Hire System."

5.1 Introduction to Video Shop System

A sample of video shop's requirements defined in [44] is selected and used in our experiment in this case study. The video shop has video club members who may hire videos. The functional requirements of video shop system concerns with how to register new video titles, how to search the video title for hiring, how to register a new member, and how to return a hired video back for each member.

According to our scheme, the informal requirements gathered should be categorized and prepared to construct a decision table. The simple way is to determine and extract all of independent events occurred. An event list, so-called 'Shall' statements, of requirements of video shop system is listed as follows:

- A new video title shall be registered into video stock.
- An existing video title shall be hired by a club member.
- A new club member shall be added into the system.
- A hired video shall be returned back into video stock.

Each event statement listed above will be considered as a rule in the target decision table and a set of preconditions and actions is about to defined in the next step.

5.1.1 Decision Table of Video Shop System

A decision table defines a set of rules to be implemented by the target software system. The developer should be able to identify a set of conditions to be satisfied and a set of actions to be performed for each rule.

Table 5.1 Decision table of the Video shop system

PreConditions	Rule1	Rule2	Rule3	Rule4
1) Video title exists		X		
2) Video title does not exist	X			
3) Member name exists		X		
4) Member name does not exist			X	
5) Hiring record exist				X
Actions				
1) Insert new video title into video stock	X			
2) Hire video title from stock		X		
3) Insert new club member			X	
4) Return the hired video title				X

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

From decision table 5.1, four decision rules are shown and each rule is described by a set of preconditions and actions. The next step is to define the primitive operations to be needed to perform the assertion of the preconditions and implement the actions. As we mentioned earlier, the primitive operations we expected would be the operations to manipulate the set of data, such as inserting, deleting, updating, and searching etc. In this case, we finally come up with 3 primitive operations – “Search”, “Store”, “Remove.” We also called these three primitive operations as “Particle” in our RPN notation.

The decision rules would be discussed in detail on how to use our primitive operations to perform the assertion and actions as follows:

- **Rule 1: Insert New Video.** This rule specifies the action of inserting a new video title into video stock. A "Search" operation should be used to search if the video title exists. If not, the new video title will be stored into the video stock using a "Store" operation.
- **Rule 2: Hire a Video.** This rule specifies the action of hiring video title from video stock. A "Search" operation should be used to search if the hirer is club member and a "Search" operation is also used to search if the video title exists. If the preconditions are satisfied. The existing video title will be hired to the club member. In order to keep the hiring record, a "Store" operation will be used to store the hiring record.
- **Rule 3: Insert New Member.** This rule specifies the action of inserting a new club member. A "Search" operation should be used to search if the club member exists. If not, the new member name will be stored into the video stock using a "Store" operation.
- **Rule 4: Return a Video.** This rule specifies the action of returning the hired video title. A "Search" operation should be used to search if the hiring record exists. The existing hiring record will deleted by using "Remove" operation.

The decision rules would be verified with the end user, if needed. At this stage, the informal requirements specification is now rewritten into a set of rules. A set of operations is identified and it is ready to go on the next step - defining a requirements particle definition for each primitive operation. We intend to keep the number of primitive operations minimized and encourage the reusability of the defined primitive operations.

5.1.2 Requirements Particle Definitions of Primitive Operations

From the previous section, we design to have 3 primitive operations for this case - “Store”, “Search”, and “Remove” operations. In this section, we present a relevant and generic requirements particle definition in Z for each primitive operation – “Particle”.

“Store” Particle Notation. The “Store” particle notation is shown in figure 5.1 and its corresponding Z requirements particle definition is listed in horizontal schema format. As seen in the requirements particle definition, the schema name is named after the particle name. The first part of the schema is the declaration of local variables – names and types. The second part of the schema is the axiom predicates that specify the union between the current set of data and the set of a new member. Moreover, the output ports are specified to forward both data and status messages to the consecutively connected particle.

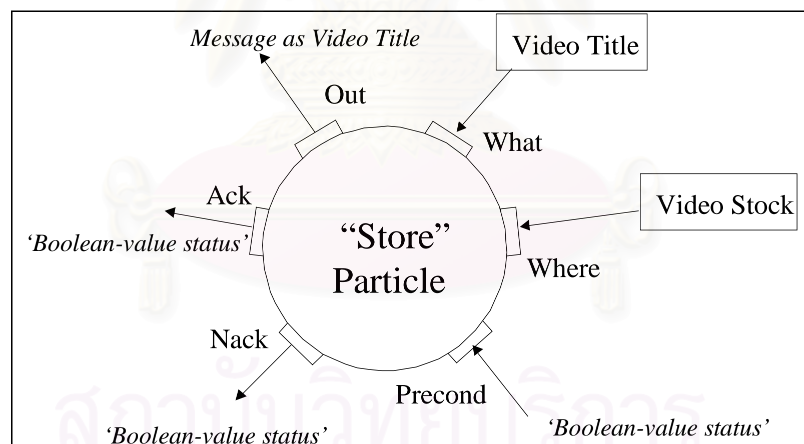


Figure 5.1 Sample of “Store” Particle Notation

Store / *[What? : What_Type;*
Where?, Where’ : Where_Type;
Ack! : Boolean;
Nack! : Boolean;
Out! : Out_type; |
Precond $\wedge$
Where’ = Where? $\cup$ *{What?}* $\wedge$

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

$Out! = What? \wedge$
$Ack! = What? \in Where' \wedge$
$Nack! = What? \notin Where']$

“Search” Particle Notation. The “Search” particle notation is shown in figure 5.2 and its corresponding Z requirements particle definition is listed in horizontal schema format. As seen in the requirements particle definition, the schema name is named after the particle name. The first part of the schema is the declaration of local variables – names and types. The second part of the schema is the axiom predicates that specify the assertion of the membership of a set. Moreover, the output ports are specified to forward both data and status messages to the consecutively connected particle.

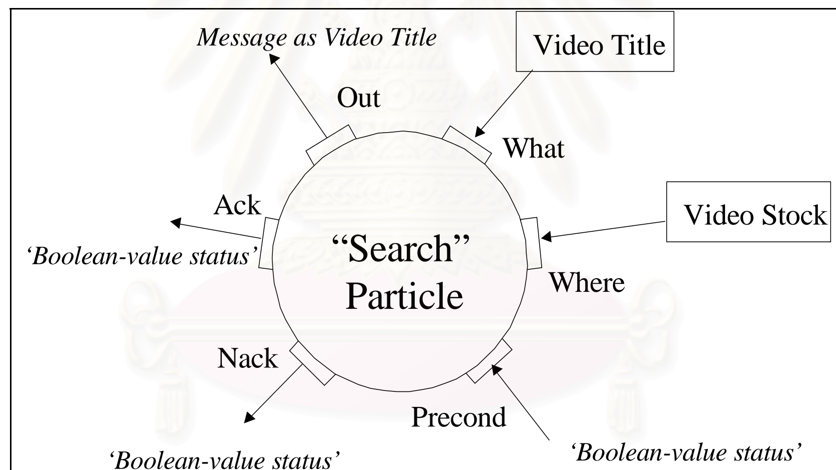


Figure 5.2 Sample of “Search” Particle Notation

$Search \,/\, [What? : What_Type;$
$Where? : Where_Type;$
$Ack! : Boolean;$
$Nack! : Boolean;$
$Out! : Out_Type; \;|$
$Precond \wedge$
$Out! = What? \wedge$
$Ack! = What? \in Where? \wedge$
$Nack! = What? \notin Where?]$

"Remove" Particle Notation. The "Remove" particle notation is shown in figure 5.3 and its corresponding Z requirements particle definition is listed in horizontal schema format. As seen in the requirements particle definition, the schema name is named after the particle name. The first part of the schema is the declaration of local variables – names and types. The second part of the schema is the axiom predicates that specify the difference between the current set of data and the set of a member. Moreover, the output ports are specified to forward both data and status messages to the consecutively connected particle.

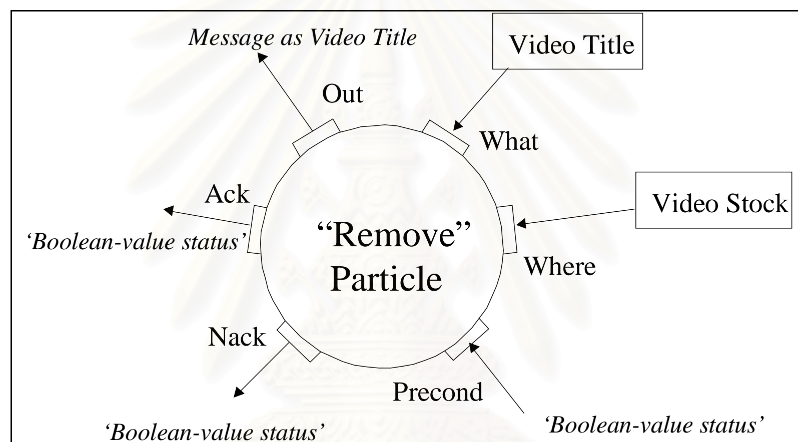


Figure 5.3 Sample of "Remove" Particle Notation

$$
\begin{array}{ll}
Remove \mid & [What? : What_Type; \\
& Where?, Where' : Where_Type; \\
& Ack! : Boolean; \\
& Nack! : Boolean; \\
& Out! : Out_Type; \mid \\
& Precond \wedge \\
& Where! = Where? \setminus \{What?\} \wedge \\
& Out! = What? \wedge \\
& Ack! = What? \notin Where' \wedge \\
& Nack! = What? \in Where']
\end{array}
$$

At this point, we now provide a set of primitive operation notations and their corresponding Z requirements particle definition to be

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

used in the next step. The decision rules will be concerned and RPNs will be drawn to represent each decision rule. Then, each Z requirements particle definition will play its role.

5.1.3 RPN for Each Decision Rule

This section presents RPN drawing for each decision rule defined according to the decision table. The Z requirements particle definitions, in this case formal definitions for “Store”, “Search”, and “Remove” are determined to specify the primitive operations in a RPN drawing. The transformation rule 1, described in previous chapter, is used to perform the substitution with the appropriate data sets into the formal definitions to construct a meaningful Z specification part. After finishing the substitution, transformation rule 2 is then used to generate the glue predicates to gather all of the partial Z specifications together. Thus, a decision rule is now formally defined.

To implement the substitution by using transformation rule 1, we need to do the inclusion of relevant schemas and renaming local variables within each Z schema uniquely. The details are shown below.

Rule 1: Insert New Video.

As shown in figure 5.4, the rule of inserting new video will search for the existing of the video title. If the video title is not existing in the current video stock, the new video title will be then inserted into the video stock. We found that data sets of “Video Title”, “Video Stock” are essentially needed to carry the video title and the video stock. Practically, in Z, given set named “VIDEOTITLE” and its power set named “VIDEOSTOCK” are specified.

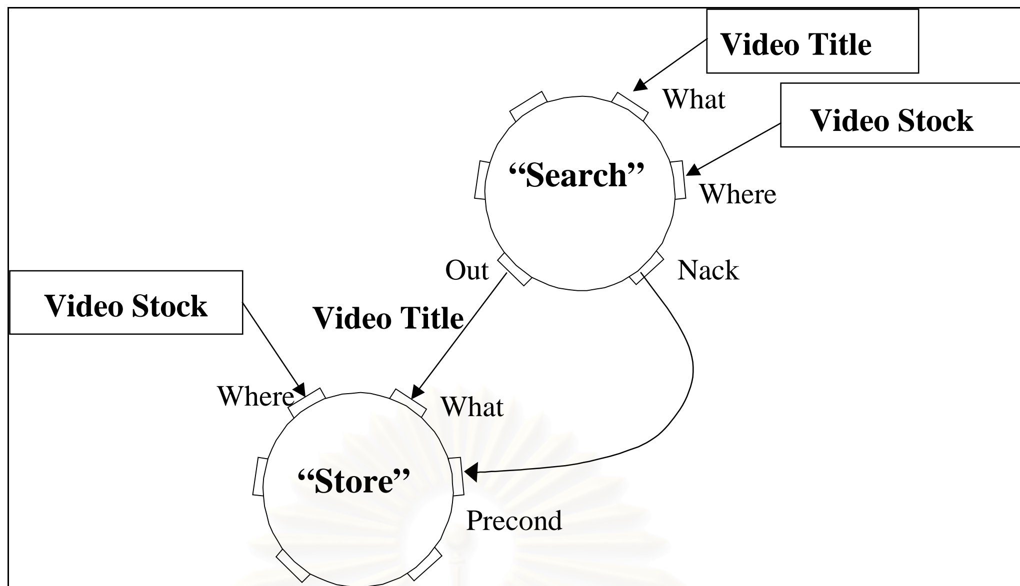


Figure 5.4 RPN Drawing for Rule1 – Insert New Video

The partial Z specification generated from the RPN is listed as follows:

The data sets are defined as follows:

[VIDEOTITLE]
VIDEOSTOCK = = *Π VIDEOTITLE*

The operation schemas and local variables are defined and renamed as follows:

Rule1Search / [VideoTitle? : VIDEOTITLE;
VideoStock? : VIDEOSTOCK;
Rule1SearchAck! : Boolean;
Rule1SearchNack! : Boolean;
Rule1SearchOut! : VIDEOTITLE |
Rule1SearchOut! = VideoTitle? $\wedge$
Rule1SearchAck! = VideoTitle? $\in$ *VideoStock?* $\wedge$
Rule1SearchNack! = VideoTitle? $\notin$ *VideoStock?]*

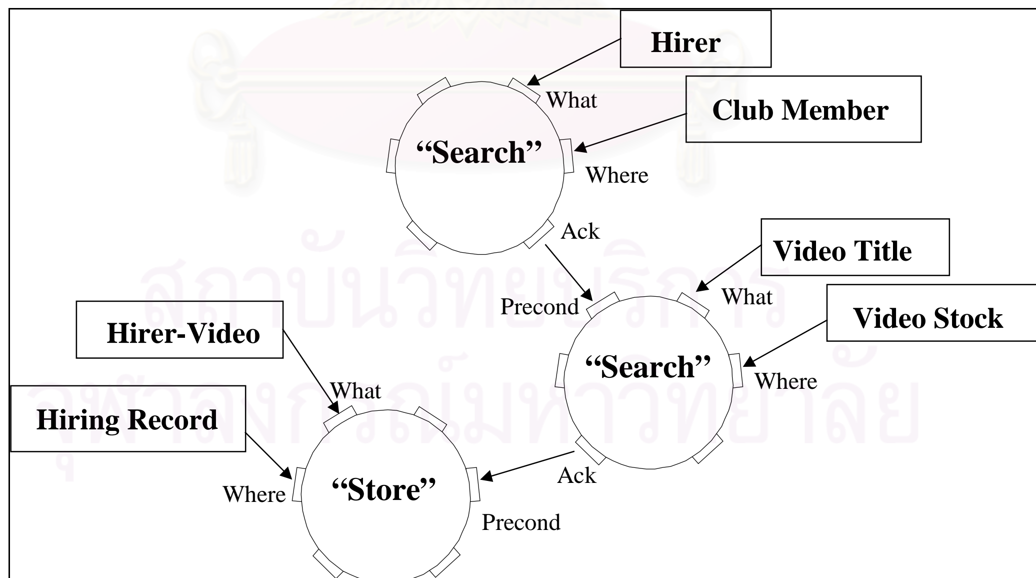
Rule1Store / [VideoTitle? : VIDEOTITLE;
VideoStock?, VideoStock' : VIDEOSTOCK;
Rule1Store Ack! : Boolean;
Rule1Store Nack! : Boolean;
Rule1Store Out! : VIDEOTITLE;
Rule1Search |

$VideoTitle? \notin VideoStock? \wedge$
$VideoStock' = VideoStock? \cup \{VideoTitle?\} \wedge$
$Rule1Store\ Out! = VideoTitle? \wedge$
$Rule1Store\ Ack! = VideoTitle? \in VideoStock' \wedge$
$Rule1Store\ Nack! = VideoTitle? \notin VideoStock']$

The glue predicate is defined (using transformation rule 2) as follows:
$Rule1 \,/\, Rule1Search \wedge Rule1Store$

Rule 2: Hire a Video.

As shown in figure 5.5, the rule of hiring a video will verify the membership of the hirer by searching the existing club member. Only club member is allowed to hire a video. Then, a search operation will check for the existing of the video title. If the video title exists in the current video stock, the relevant hiring record will be stored. We found that data sets of “Video Title”, “Video Stock”, “Hirer”, “Club Member”, Hirer-Video”, and “Hiring Record” are essentially needed. Practically, in Z, given sets named “VIDEOTITLE”, “HIRER”, “HIRERVIDEO” and their power sets named “VIDEOSTOCK”, “CLUBMEMBER”, and “HIRINGRECORD” are specified respectively.



สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

Figure 5.5 RPN Drawing for Rule2 – Hire a Video

The Z specification generated from the RPN is listed as follows:

The data sets are specified as follows:

[VIDEOTITLE]
VIDEOSTOCK = = Π *VIDEOTITLE*
[HIRER]
CLUBMEMBER = = Π *HIRER*
[HIRERVIDEO]
HIRINGRECORD = = Π *HIRERVIDEO*

The operation schemas are specified as follows:

Rule2Search1 / [Hirer? : HIRER;
ClubMember? : CLUBMEMBER;
Rule2Search1Ack! : Boolean;
Rule2Search1Nack! : Boolean;
Rule2Search1Out! : HIRER |
Rule2Search1Out! = Hirer? $\wedge$
Rule2Search1Ack! = Hirer? $\in$ *ClubMember?* $\wedge$
Rule2Search1Nack! = Hirer? $\notin$ *ClubMember?]*

Rule2Search2 / [VideoTitle? : VIDEOTITLE;
VideoStock? : VIDEOSTOCK;
Rule2Search2Ack! : Boolean;
Rule2Search2Nack! : Boolean;
Rule2Search2Out! : VIDEOTITLE;
Rule2Search1 |
Hirer? $\in$ *ClubMember?*$\wedge$
Rule2Search2Out! = VideoTitle? $\wedge$
Rule2Search2Ack! = VideoTitle? $\in$ *VideoStock?* $\wedge$
Rule2Search2Nack! = VideoTitle? $\notin$ *VideoStock?]*

Rule2Store / [HirerVideo? : HIRERVIDEO;
HiringRecord?, HiringRecord' : HIRINGRECORD;
Rule2Store Ack! : Boolean;
Rule2Store Nack! : Boolean;
Rule2Store Out! : HIRERVIDEO;
Rule2Search2 |
VideoTitle? $\in$ *VideoStock?* $\wedge$

$HiringRecord' = HiringRecord? \cup \{HirerVideo?\} \wedge$
$Rule2Store\ Out! = HirerVideo? \wedge$
$Rule2Store\ Ack! = HirerVideo? \in HiringRecord' \wedge$
$Rule2Store\ Nack! = HirerVideo? \notin HiringRecord']$

The glue predicate is defined as follows:

$Rule2 \,/\, Rule2Search1 \wedge (\, Rule2Search2 \wedge Rule2Store\,)$

Rule 3: Insert New Member.

As shown in figure 5.6, the rule of inserting new club member will search for the existing of the hirer. If the hirer is not existing in the current set of club members, the new hirer will be then inserted into the club member set. We found that data sets of “Hirer”, “Club Member” are essentially needed. Practically, in Z, given set named “HIRER” and its power set named “CLUBMEMBER” are specified.

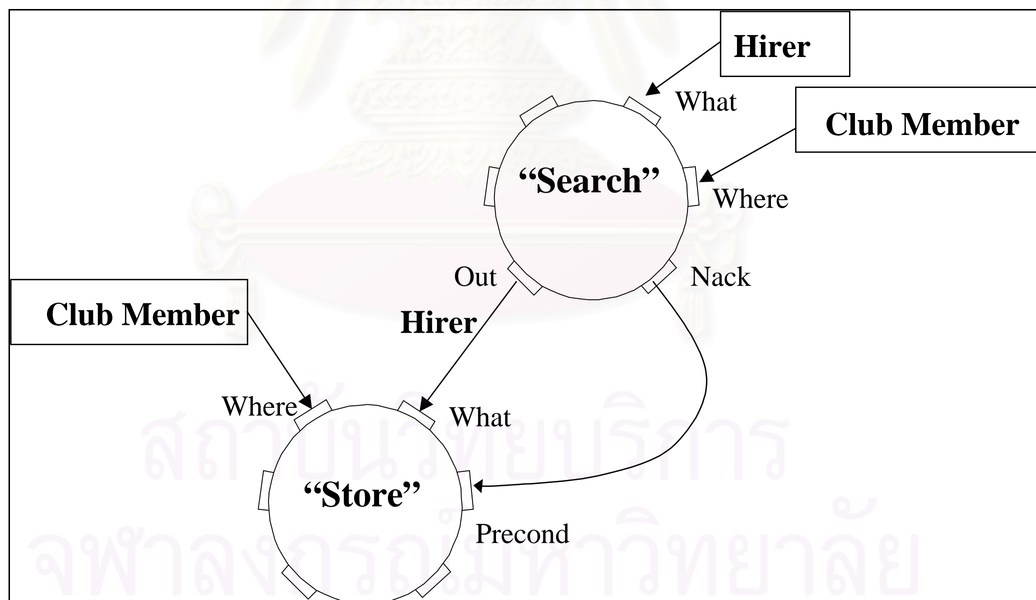


Figure 5.6 RPN Drawing for Rule 3 – Insert New Member

The partial Z specification generated from the RPN is listed as follows:

The data sets are defined as follows:

$$\begin{array}{l} [HIRER] \\ CLUBMEMBER == \Pi HIRER \end{array}$$

The operation schemas and local variables are defined and renamed as follows:

$$\begin{array}{ll} Rule3Search \,/\, & [Hirer? : HIRER; \\ & ClubMember? : CLUBMEMBER; \\ & Rule3SearchAck! : Boolean; \\ & Rule3SearchNack! : Boolean; \\ & Rule3SearchOut! : HIRER; \mid \\ & Rule3SearchOut! = Hirer? \wedge \\ & Rule3SearchAck! = Hirer? \in ClubMember? \wedge \\ & Rule3SearchNack! = Hirer? \notin ClubMember?] \end{array}$$

$$\begin{array}{ll} Rule3Store \,/\, & [Hirer? : HIRER; \\ & ClubMember?, ClubMember' : CLUBMEMBER; \\ & Rule3StoreAck! : Boolean; \\ & Rule3StoreNack! : Boolean; \\ & Rule3StoreOut! : HIRER; \\ & Rule3Search \mid \\ & Hirer? \notin ClubMember? \wedge \\ & ClubMember' = ClubMember? \cup \{Hirer?\} \wedge \\ & Rule3StoreOut! = Hirer? \wedge \\ & Rule3StoreAck! = Hirer? \in ClubMember' \wedge \\ & Rule3StoreNack! = Hirer? \notin ClubMember'] \end{array}$$

The glue predicate is defined as follows:

$$Rule3 \,/\, Rule3Search \wedge Rule3Store$$

Rule 4: Return a Video.

As shown in figure 5.7, the rule of returning a video will search for the existing of the hiring record. If the hiring record exists in the current hiring record, the hiring record will be then removed from the hiring record set. We found that data sets of “Hirer-Video”, “Hiring Record” are essentially needed. Practically, in Z, given set named “HIRERVIDEO” and its power set named “HIRING RECORD” are specified.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

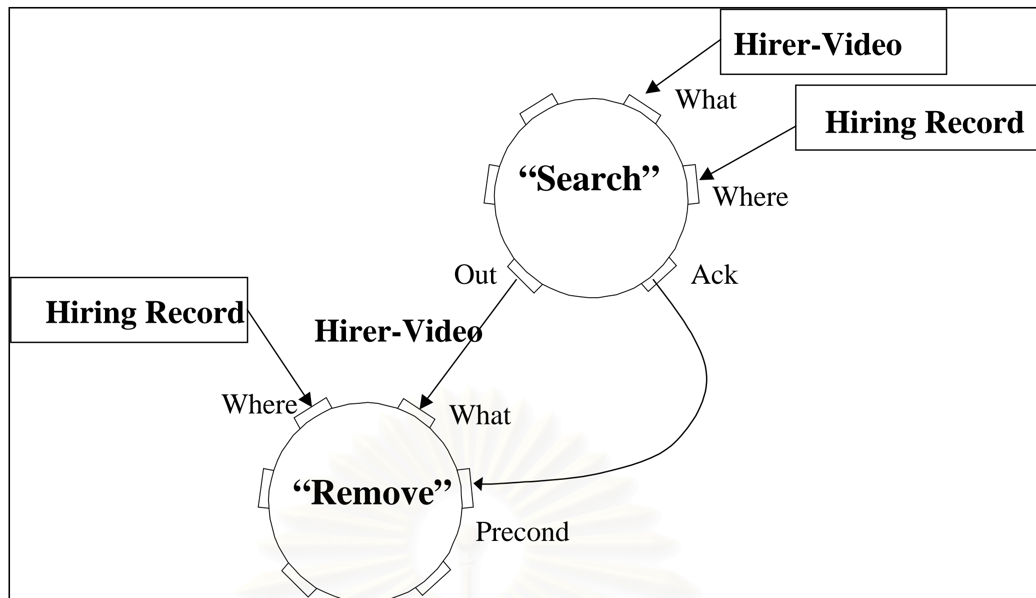


Figure 5.7 RPN Drawing for Rule 4 – Return a Video

The partial Z specification generated from the RPN is listed as follows:

The data sets are defined as follows:

[HIRERVIDEO]
HIRINGRECORD = = Π HIRERVIDEO

The operation schemas and local variables are defined and renamed as follows:

Rule4Search / [HirerVideo? : HIRERVIDEO;
HiringRecord? : HIRINGRECORD;
Rule4SearchAck! : Boolean;
Rule4SearchNack! : Boolean;
Rule4SearchOut! : HIRER;
Rule4SearchPrecond? : Boolean
| Rule4SearchPrecond? = TRUE ∧
Rule4SearchOut! = HirerVideo? ∧
Rule4SearchAck! = HirerVideo? ∈ HiringRecord? ∧
Rule4SearchNack! = HirerVideo? ∉ HiringRecord?]

Rule4Remove / [HirerVideo? : HIRERVIDEO;
HiringRecord?, HiringRecord' : HIRINGRECORD;
Rule4RemoveAck! : Boolean;

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

$Rule4RemoveNack! : Boolean;$
$Rule4RemoveOut! : HIRERVIDEO;$
$Rule4Search \mid$
$HirerVideo? \in HiringRecord? \wedge$
$HiringRecord' = HiringRecord? \setminus \{HirerVideo?\} \wedge$
$Rule4RemoveOut! = HirerVideo? \wedge$
$Rule4RemoveAck! = HirerVideo? \notin HiringRecord' \wedge$
$Rule4RemoveNack! = HirerVideo? \in HiringRecord']$

The glue predicate is defined as follows:

$Rule4 \,/\, Rule4Search \wedge Rule4Remove$

5.1.4 Final specifications

The final specification of video shop system, VDOSHOPSPEC, is formally concluded from all of possible decision rules. The transformation rule 3 is referred to do this gathering by using disjunctive approach.

$VDOSHOPSPEC \,/\, Rule1 \vee Rule2 \vee Rule3 \vee Rule4$

We list the complete Z specifications of video shop system in appendix A.1. The final version of the formal specifications would be used as the first draft to describe the software system.

สถาบันวิทยบริการ

5.1.5 Conclusions on Video Shop System

จุฬาลงกรณ์มหาวิทยาลัย

Since the case study of video shop system is not too complicate, we could gradually walkthrough the steps of our specification synthesis scheme in detail. The data sets of the system, such as Video Title, Hirer, etc. are directly viewed as the set of elements. The insertion is considered as the union of two sets and the deletion is the difference of two sets, in set theory.

As reported in [52], we conducted a workshop of developing software requirements specification of this case. More than 80 attendants (undergraduate and

graduate students) with experience in drawing data flow diagrams are gathered. Average time to accomplish the specification procedure is 50 minutes and more than 90% of the attendants produce the complete RPNs. The final formal specifications are consequently mapped from their RPNs without any major complication.

In the next case study, we would describe how to cope with the data entity and relations between them in the database application. Our RPN notations will be applied to construct composite operations on database structures, which are formally defined in [39].

5.2 Introduction to Van Hire System

The objective of this case study is to demonstrate the usage of RPN on database application specification. Refer to [39], we proposed the transformation rules to generate a set of Z schemas from tabular form of entities and relations for a van hire system. We now extend the work to cover the specification of the composite operations using RPN notations and report in [45]. The contents of the reports will be described in this section.

In this section, we begin with the brief overview of the related formal definitions on how we define the entity and relation tables from the ER diagrams and how to define the structural constraints among entities and relations. We then concentrate on the usage of RPN to define the composite operations, that is the main concerns for this dissertation.

Here is the informal description of a van hire system. A van hire system is to be developed for a garage, which runs a van hire business. Each van is classified into only one van class. A van in the garage will be in service no more than 10 years. A customer begins to make a booking for a particular van class and an available van in the specified category will be assigned. The customer will check out the van at the garage's hire center and finally return the van at the end of hire period. The historical data is not needed for this system.

5.2.1 The Entity Relationship Diagram of Van Hire System

In figure 5.8, the entity relationship diagram shows four entities and four relations among entities. The entities are VAN, VANCLASS, CUSTOMER, and BOOKING. The relations are ISIN, ISGIVEN, RESERVE, MAKES. The ISIN is the many-to-one relation between VAN and VANCLASS. The RESERVE is the many-to-one relation between BOOKING and VANCLASS. The MAKES is the one-to-many relation between CUSTOMER and BOOKING. The ISGIVEN is the one-to-one relation between VAN and BOOKING. The data dictionary table shown in figure 5.10 and figure 5.11 include all of the attribute names, attribute types, attribute constraints, the selections of primary keys and foreign keys and relationship among entities.

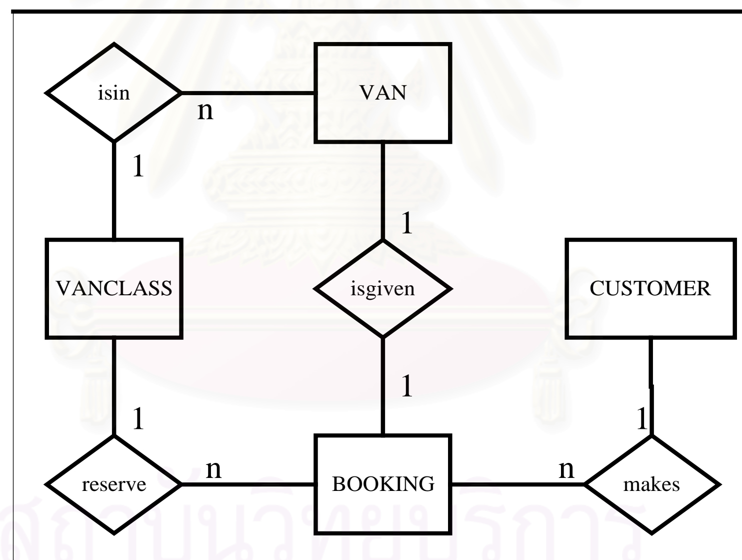


Figure 5.8: ER Diagram of Van Hire System [39]

5.2.2 Formal Specification Synthesis Scheme

In [39], we utilize the conceptual model of the entity relationship diagram and their data dictionaries, which are commonly prepared by software analyst and designer to be our original inputs. Now, we propose the extended part of the

specification and the overview of the new scheme of formal specification synthesis is shown in figure 5.9. The final specifications include both structural property and behavioral property of relational data model. The structural property is specified by Z data schemas – entity schemas, entity extension schemas, and relationship schemas. Meanwhile, the behavioral property is specified by Z operation schemas – primitive operation schemas and new composite operation schemas.

Table 5.2 Data Dictionary for all Entity Names of Van Hire System

Entity Name: VAN

Attribute name	**Attribute type**	**Key**	**Constraint**
RegistrationNo	string	PK	
Model	string		
vanclassFK	CLASSNAME	FK(VANCLASS-className)	
usedPeriod	integer		usedPeriod ノ 10

Entity Name: VANCLASS

Attribute name	**Attribute type**	**Key**	**Constraint**
className	CLASSNAME	PK	
fuelType	FUELTYPE		

Entity Name: BOOKING

Attribute name	**Attribute type**	**Key**	**Constraint**
bookingCode	string	PK	
startDate	date		
endDate	date		
customerID	string	FK(CUSTOMER-customerId)	
class	CLASSNAME	FK(VANCLASS-className)	
vanNo	string	FK(VAN-registerNo)	

Entity Name: CUSTOMER

Attribute name	**Attribute type**	**Key**	**Constraint**
CustomerId	string	PK	
Name	string		
Address	string		

Table 5.3 Relationship Dictionary of Van Hire System

Relation Name	**Entity1**	**Entity2**	**Cardinality ratio**	**Constraint**
isin	VAN	VANCLASS	n-1	
reserve	BOOKING	VANCLASS	n-1	
makes	CUSTOMER	BOOKING	1-n	
isgiven	BOOKING	VAN	1-1	

5.2.2.1 Formal Specification of Structural Property

The transformation rules of generating formal specifications of the structural property of relational data model is proposed in [39]. The entity relationship diagram and its data dictionary, commonly prepared during the early stage of software development, are used as original definitions. An entity relationship diagram is transformed into three categories of schemas in Z: Entity schema, Entity Extension schema, and Relationship schema. The Entity schema represents all of the attributes and types in an entity. In the meantime, the Entity Extension schema defines the constraints of structural property such as primary key, foreign key, as well as the integrity constraints. The Relationship schema defines the relations between entities along with the cardinality ratio - one-to-one, one-to-many, and many-to-one relationship.

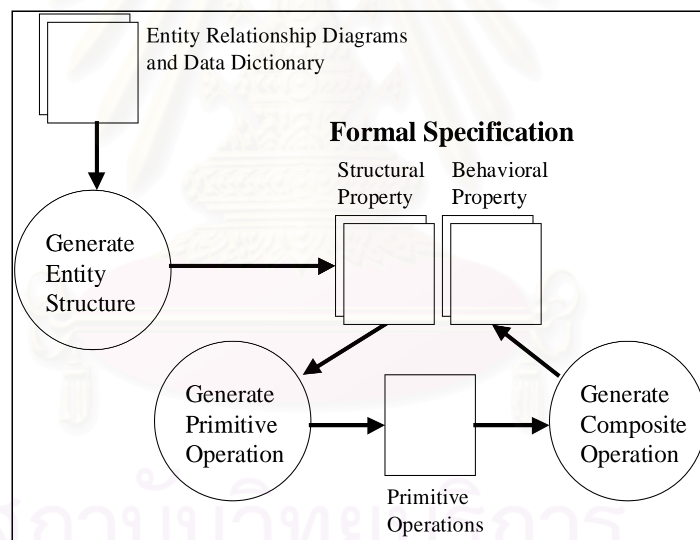


Figure 5.9 Formal Specification Synthesis Scheme for Database Application

5.2.2.2 Composition of Primitive Operations

This section describes how to compose the primitive operations, such as insertion, deletion, updating, etc. into a composite operation. The requirements particle network is used to construct a new composite operation.

5.2.2.3 Constructing a New Composite Operation

A new composite operation is a set of primitive operations that are tightly coupled and is defined by a RPN. Each primitive operation can be justified and fit as a particle node in the RPN.

5.2.2.4 Composition Operation Generating Rules

Based on our formal specification synthesis scheme, a composite formal specification is generated using a RPN as follows:

- Select a set of primitive operations *OPi* that provide the relevant features.
- Compose each primitive operation from top-down style.
- Specify the operation for the *Ack* path, if the current operation is successfully performed.
- Specify the operation for the *Nack* path, if the current is fail.
- Specify the Data entity for each operation, if needed.
- Define the rest of the operations.

For example, a RPN called COMPO1 is designed as shown in figure 5.10 and *OP1*, …, *OP6* are our primitive operations such as insert, delete, etc. If the operation *OP1* is successfully performed then pursue the operation *OP2* and if not, invoke *OP3*.

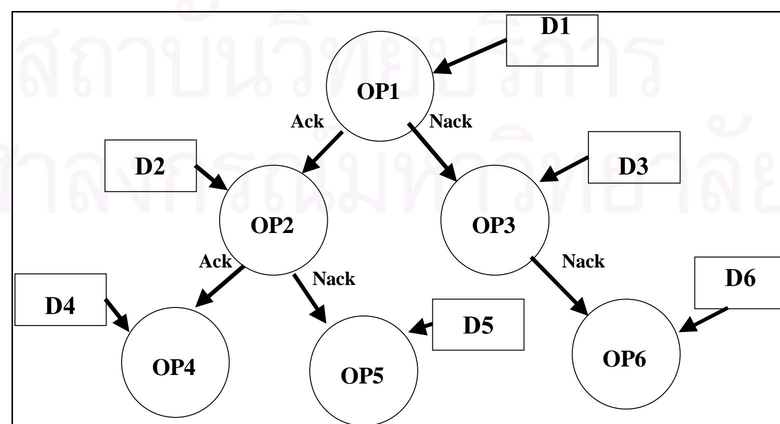


Figure 5.10 A Composite Operation ‘*COMPO1*’

The composite operation '*COMPO1*' is generated as follows:

$$COMPO1 \;/\; (OP1 \wedge OP2 \wedge OP4)\ \varpi$$
$$(OP1 \wedge \neg OP2 \wedge OP5)\ \varpi\ (\neg OP1 \wedge \neg OP3 \wedge OP6)$$

5.2.3 Samples of Composite Operations for Van Hire System

In this section, we give the samples of two composite operations - booking and returning a van. The RPN approach is slightly modified so that we could reuse the primitive operations of inserting, deleting, and updating given from [39]. By using them, we do not need to add more Z notations for maintaining the structural constraints of the entities and relations. Moreover, the cascading inserting and deleting are still implicitly supported.

The following Z specifications will give a idea of how the primitive operations called "InsertBooking" and "DeleteBooking" work. Both of operations will be invoked in our RPN formal definitions.

[DATE, FUELTYPE, STRING]

Vanclass / [ClassName : STRING;
FuelType : FUELTYPE]

Van / [Reg : STRING;
Model : STRING;
Class : STRING]

Customer / [CustId : STRING;
CustName : STRING;
CustAddr : STRING]

Booking / [BookId : STRING;
StartDate : DATE;
EndDate : DATE;
CustId : STRING;
Class : STRING;
VanNo : STRING]

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

VanclassExt / [VanclassSet : Φ Vanclass |
A Vanclass1, Vanclass2 : Vanclass |
Vanclass1 ∈ VanclassSet ∧
Vanclass2 ∈ VanclassSet ∧
Vanclass1 ⌊ Vanclass2 •
Vanclass1.ClassName ⌊ Vanclass2.ClassName]

VanExt / [VanSet : Φ Van |
A Van1, Van2 : Van |
Van1 ∈ VanSet ∧
Van2 ∈ VanSet ∧
Van1 ⌊ Van2 •
Van1.Reg ⌊ Van2.Reg ∧
A VanVar : Van •
(∃ VanclassVar : Vanclass •
VanVar.Class = VanclassVar.ClassName)]

CustomerExt / [CustomerSet : Φ Customer |
A Customer1, Customer2 : Customer |
Customer1 ∈ CustomerSet ∧
Customer2 ∈ CustomerSet ∧
Customer1 ⌊ Customer2 •
Customer1.CustId ⌊ Customer2.CustId]

BookingExt / [BookingSet : Φ Booking |
A Booking1, Booking2 : Booking |
Booking1 ∈ BookingSet ∧
Booking2 ∈ BookingSet ∧
Booking1 ⌊ Booking2 •
Booking1.BookId ⌊ Booking2.BookId ∧
A BookingVar : Booking •
(∃ VanVar : Van •
BookingVar.VanNo = VanVar.Reg)
A BookingVar : Booking •
(∃ VanclassVar : Vanclass •
BookingVar.VanNo = VanclassVar.ClassName)
A BookingVar : Booking •
(∃ CustomerVar : Customer •
BookingVar.CustId = CustomerVar.CustId)]

InsertBooking / [Δ BookingExt;
NewValue? : Booking
InsertCustomer
InsertVanclass
InsertVan |
BookingSet' = BookingSet ∪ {NewValue?}]

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

DeleteBooking / *[Δ BookingExt;*
dBooking? : Booking |
BookingSet' = BookingSet \ {dBooking?}]

UpdateBooking / *[DeleteBooking; InsertBooking]*

5.2.3.1 Samples of Decision Table for Van Hire System

A decision table is prepared to describe the conditions and actions of van hire system, especially for composite operations.

Table 5.4 Decision Table of the Van Hire System

PreConditions	Rule1	Rule2
1) Van exists	X	
2) Hiring record exist		X
Actions		
1) Insert the hiring record	X	
2) Delete the existing hiring record		X

From decision table 5.4, two decision rules are shown and each rule is described by a set of preconditions and actions. The next step is to define the primitive operations to be needed to perform the assertion of the preconditions and implement the actions.

The decision rules are listed below.

- **Rule 1: Book a Van.** This rule specifies the action of booking a van in van hire system. A "Search" operation should be used to search if the van exists. If the van exists, the booking record will be recorded using a "InsBooking" operation.
- **Rule 2: Return a Van.** This rule specifies the action of returning a van back to van hire system. A "Search" operation should be used to search if

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

the booking record exists. If it exists, the booking record will be deleted using a “DelBooking” operation.

5.2.3.2 Samples of Requirements Particle Definitions

From the previous section, there are 3 primitive operations for this case - “Search”, “InsBooking”, and “DelBooking” operations. In this section, we present a relevant and generic requirements particle definition in Z for each primitive operation – “Particle”.

“Search” Particle Notation. The “Search” particle notation is shown in figure 5.11 and its corresponding Z requirements particle definition is listed in horizontal schema format. As seen in the requirements particle definition, the schema name is named after the particle name. The first part of the schema is the declaration of local variables – names and types. The second part of the schema is the axiom predicates that specify the assertion of the membership of a set. Moreover, the output ports are specified to forward both data and status messages to the consecutively connected particle.

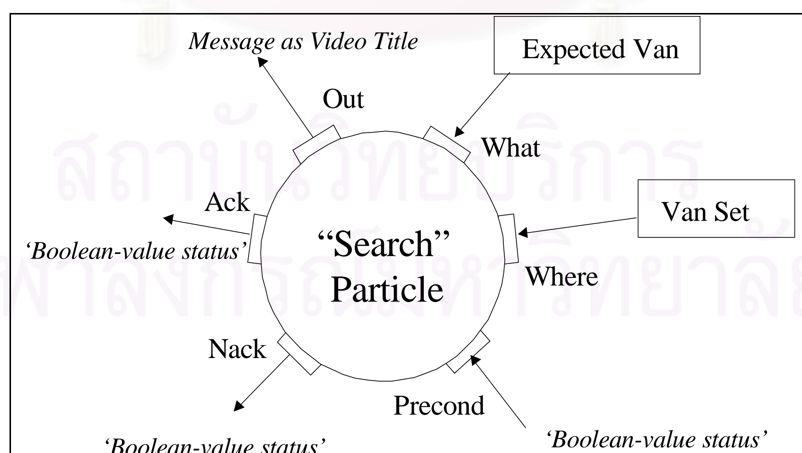


Figure 5.11 Sample of “Search” Particle Notation

Search / *[What? : What_Type;*
Where? : Where_Type;
Ack! : Boolean;
Nack! : Boolean;
Out! : Out_Type |
Precond ∧
Out! = What? ∧
Ack! = What? ∈ *Where?* ∧
Nack! = What? ∉ *Where?]*

“InsBooking” Particle Notation. The “InsBooking” particle notation is shown in figure 5.12 and its corresponding Z requirements particle definition is listed in horizontal schema format. As seen in the requirements particle definition, the schema name is named after the particle name. The first part of the schema is the declaration of local variables – names and types. The “InsertBooking” schema generated from [39] will be included here to manipulate the state schema “Booking.” Moreover, the output ports are specified to forward both data and status messages to the consecutively connected particle.

/

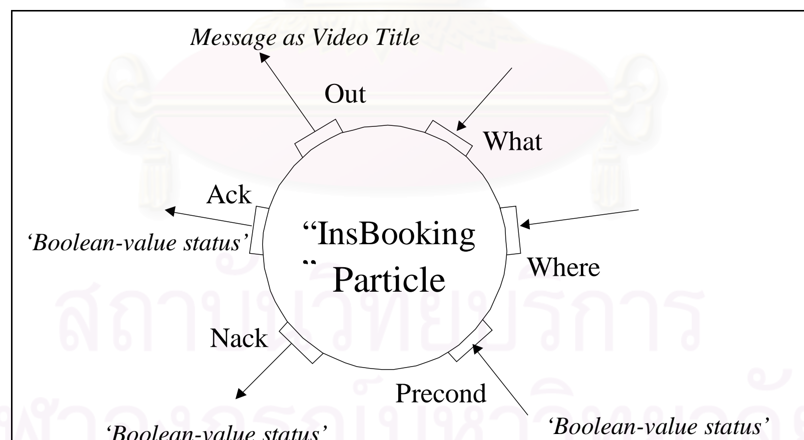


Figure 5.12 Sample of “InsBooking” Particle Notation

InsBooking / *[Ack! : Boolean;*
Nack! : Boolean;
InsertBooking |
Precond ∧
Ack! = Precond ∧
Nack! = !Precond]

"DelBooking" Particle Notation. The "DelBooking" particle notation is shown in figure 5.13 and its corresponding Z requirements particle definition is listed in horizontal schema format. As seen in the requirements particle definition, the schema name is named after the particle name. The first part of the schema is the declaration of local variables – names and types. The "DeleteBooking" schema generated from [39] will be included here to manipulate the state schema "Booking." Moreover, the output ports are specified to forward both data and status messages to the consecutively connected particle.

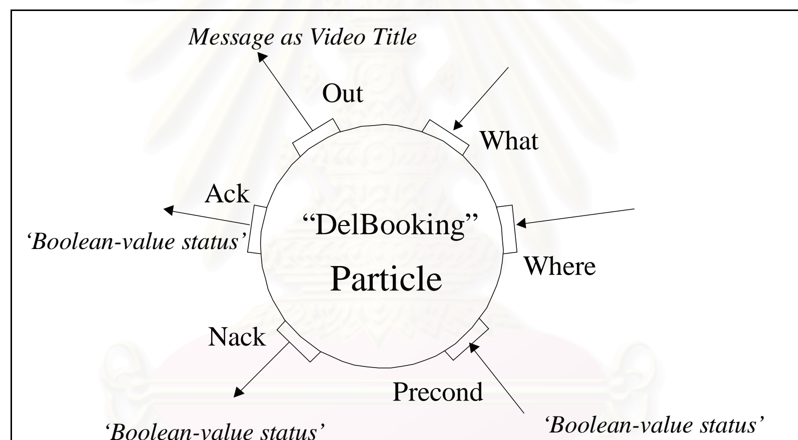


Figure 5.13 Sample of "DelBooking" Particle Notation

DelBooking ≙ [Ack! : Boolean;
Nack! : Boolean;
DeleteBooking |
Precond $\wedge$
Ack! = Precond $\wedge$
Nack! = !Precond]

5.2.3.3 Samples of RPNs

This section presents RPN drawing for the sample decision rule defined previously. The Z requirements particle definitions, in this case formal definitions for "Search", "InsBooking", and "DelBooking" are determined to specify the primitive operations in a RPN drawing. Similarly, the transformation rule 1 and transformation rule 2 are then used to generate the partial Z specifications and their glue predicates. To implement the substitution by using transformation rule 1, we need to do the inclusion of relevant schemas and renaming local variables within each Z schema uniquely. The details are shown below.

Rule 1: Book a Van.

As shown in figure 5.14, the rule of booking a van will search for the existing of the van. If the van exists, the new booking record will be then recorded into the booking set. We found that data sets of "Van", "Van Set" are essentially needed. Practically, in Z, power set named "VANSET" are specified.

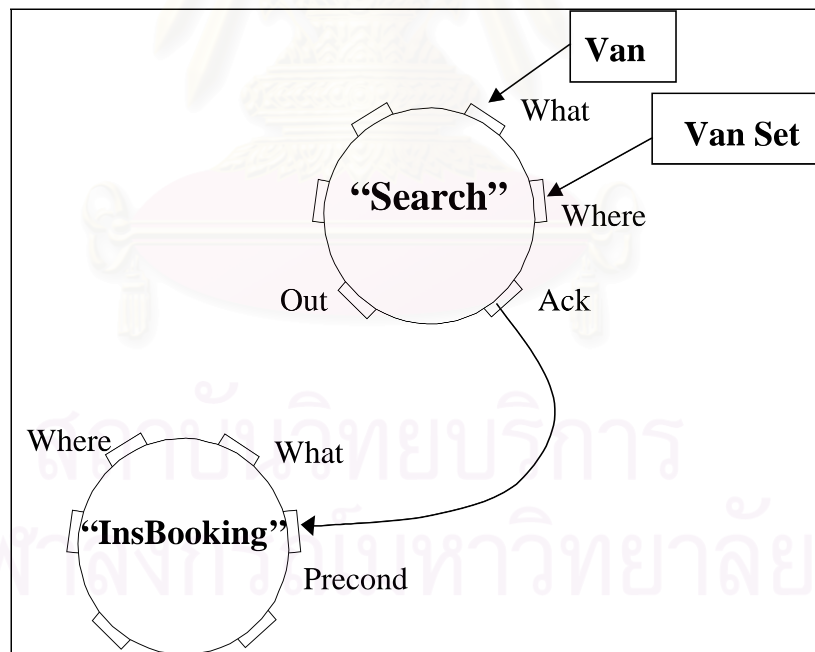


Figure 5.14 RPN Drawing for Rule1 – Book a Van

The partial Z specification generated from the RPN is listed as follows:

The data sets are defined as follows:

VANSET = = Π VAN

The operation schemas and local variables are defined and renamed as follows:

```
Rule1Search / [Van? : VAN;
                VanSet? : VANSET;
                Rule1SearchAck! : Boolean;
                Rule1SearchNack! : Boolean;
                Rule1SearchOut! : VAN |
                Rule1SearchOut! = Van? ∧
                Rule1SearchAck! = Van? ∈ VanSet? ∧
                Rule1SearchNack! = Van? ∉ VanSet?]

Rule1InsBooking / [Rule1InsBookingAck! : Boolean;
                    Rule1InsBookingNack! : Boolean;
                    InsertBooking
                    Rule1Search |
                    Van? ∈ VanSet? ∧
                    Rule1InsBookingAck! = (Van? ∈ VanSet?)∧
                    Rule1InsBookingNack! = !(Van? ∈ VanSet?)]
```

The glue predicate is defined (using transformation rule 2) as follows:

Rule1 / Rule1Search ∧ Rule1InsBooking

Rule 2: Return a Van.

As shown in figure 5.15, the rule of returning a van will search for the existing of the booking set. If the booking exists, the booking record will be then deleted from the booking set. We found that data sets of “Booking”, “Booking Set” are essentially needed. Practically, in Z, power set named “BOOKINGSET” are specified.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

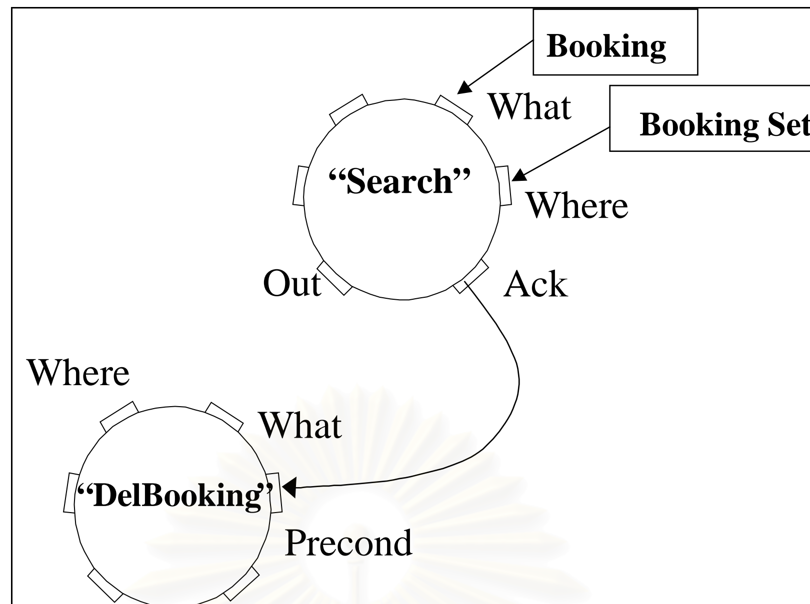


Figure 5.15 RPN Drawing for Rule1 – Return a Van

The partial Z specification generated from the RPN is listed as follows:

The data sets are defined as follows:

BOOKINGSET = = Π*BOOKING*

The operation schemas and local variables are defined and renamed as follows:

Rule2Search / *[Booking? : BOOKING;*
BookingSet? : BOOKINGSET;
Rule2SearchAck! : Boolean;
Rule2SearchNack! : Boolean;
Rule2SearchOut! : BOOKING |
Rule2SearchOut! = Booking? $\wedge$
Rule2SearchAck! = Booking? $\in$ *BookingSet?* $\wedge$
Rule2SearchNack! = Booking? $\notin$ *BookingSet?]*

Rule2DelBooking / *[Rule2DelBookingAck! : Boolean;*
Rule2DelBookingNack! : Boolean;
DeleteBooking
Rule2Search |
Booking? $\in$ *BookingSet?* $\wedge$

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

$$Rule2DelBookingAck! = (Booking? \in BookingSet?) \wedge$$
$$Rule2DelBookingNack! = !(Booking? \in BookingSet?)]$$

The glue predicate is defined (using transformation rule 2) as follows:

$$Rule2 \,/\, Rule2Search \wedge Rule2DelBooking$$

5.2.4 Conclusions on Van Hire System

This case study is to extend the work of [39]. A systematic scheme of formal specification synthesis for relational data model has been proposed in [39]. Starting from the commonly used entity relationship diagram and its related data dictionary, we propose a set of transformation rules to implement our knowledge on how to interpret entity relationship model into formal model. The formal specification of both structural and behavioral property of relational data model is produced with consistent to the original requirements model. Without any given functional information, the necessary primitive operations such as insertion, deletion, and updating are generated. All of the primitive operations mentioned perform in the cascade manners so that the referential integrity among relations is guaranteed.

In this case study, we now propose a composition technique to construct the composite operations from primitive ones. The RPN is easy and simple enough for the naive users of formal methods.

From the viewpoint of the software analyst and designer, the bringing together of the graphical diagrams and text elements of the relational model of data in Z specification helps to identify the omissions or errors in the structured analysis. Writing the formal specification ensures that the constraints, conditions, and definitions of structural property are covered. Thus, the investigation of the requirements and design specifications can be conducted using mathematical proofs.

CHAPTER 6

CONCLUSIONS AND FUTURE WORKS

This chapter concludes the comparison of our work to the related works. We summarize the primary contributions of our research and review the central results of our scheme of formal software specification synthesis. We also conclude the areas of the future works.

6.1 Comparison to the Related Works

Several researches propose the scheme of transformation from structural analysis methods to formal model. For example, the draft technique of formalizing an entity relationship diagram as Z state schemas is proposed by [46]. The notations used are the structured analysis method. The use of formal notations in the context of structured analysis is seen as valuable and the production of Z from the entity relationship diagrams is possible. In another case, the ADISSA method [47] expresses requirements using a transaction-oriented refinement of Structured Systems Analysis. The control part of ADISSA transactions are transformed into formal specification called FSM transaction using a set of rules. The design stage decomposes the FSM transaction into simpler transactions and implements them with a set of finite-state machines. Consistency between the formal specification and the result of the design is proved. The NDRASS system [6] also proposes NDRDL language based on the methods of structured analysis. The formal specification is generated by transforming the entity relationship diagram, data flow diagram, and control flow diagram with the related data dictionaries.

Comparing to the researches that develop rules from the structured analysis methods, our approach is different to the others in the following aspects:

- Most of the other approaches require complete information of both data and function definitions. The consistency between data model and functional model has to be checked. For example, a consistency checking approach between entity relationship diagram (data model) and process action diagram is proposed in [48]. The functions may refer to the undefined data entity or some data entities may be unused. Our approach provides the automatic generation of the primitive operations on each data entity, which is tightly coupled with all of the available data structure, as similar to an abstract data type.

- Whenever the complicated actions are required, a new composite operation can be constructed by reusing the previous generated primitive operations. The requirements particle network is easy for a naive user.

6.2 Contributions

Following are the primary contributions of our research to the formal specification area:

Explicit Method for Formal Specification Synthesis. Software developer will be provided with our explicit method for formal specification synthesis. The explicit method exploits the conventional techniques that most of the developers are familiar with, such as using decision table to capture the preconditions and actions of each decision rule. In our approach, a formal specification is constructed according to the defined decision rules. The steps of our scheme are concluded as follows:

- Rewrite the textual and formatted descriptions of the user requirements into a decision table.
- Define the primitive operation needed.
- Define the formal requirements particle definition for each primitive operation defined in the previous step - Z specification are focused in our studies.
- Draw a RPN for each decision rules in the decision table.

- Apply the transformation rules to convert RPN into a set of Z specifications.
- Combine the final formal specifications of all decision rules.

Graphical Notations RPN. In this research, we provide a novel graphical notations, called "Requirements Particle Network" – RPN [52]. In chapter 3, we describe both structural and behavioral aspects of RPN. We intend to retain the efficiency and usability of the visual diagrams to capture the requirements of each primitive event in the software system. By providing a set of operations, the developer can select and compose a graphical network of a decision rule. As we mentioned earlier, the operations in RPN should be reused to keep the minimum number of the predefined operations. The RPN drawings are gathered and transformed into a set of Z specifications using our transformation rules.

Transformation Rules to Synthesize Z Specifications. In our approach, we propose a set of transformation rules to synthesize Z specifications. In chapter 4, we propose the transformation rule 1 for transform each primitive operation into partial Z specifications by substitution mechanism. The transformation rule 2 will generate the glue predicates to compose all of the partial Z specifications into a decision rule and the transformation rule 3 will finalize the Z specifications by combine all of the specifications for each decision rule together using disjunctive approach.

6.3 Future Works

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

There are several future works that we consider important and recommend to be the next steps of this research.

6.3.1 Building the Common RPN Particle Library

Among the variety of requirements and needs, it is very useful to investigate and define more relevant requirements particles of the software systems for business information system. A set of particles to perform calculation is required as well. In addition, the reuse features of several common requirements particle networks should be considered.

6.3.2 Applying the RPN for Other Formal Specification Languages

In this research, we select Z specification language in our implementation. Z schema provides the features of reusability – schema inclusion, substitutions, etc. We successfully define the formal requirements particle definition of a primitive operation shown in the case studies. However, the RPN should be explored and modified to apply with the other formal specification languages, such as VDM, CafeOBJ, etc., in order to support more languages and to be more practical. At the moment, we continue our works on CafeOBJ – a algebraic specification language. The RPN needs to be modified to cope with the syntax structure of CafeOBJ language.

6.3.3 Extending the Time Synchronization Notations

The graphical notations of RPN has no specific notations to support time synchronization of the software specification. At the moment, RPN supports only sequential system. The extra notation should be extended into the original RPN to support the synchronization among operations. An operation may be activated by more than one acknowledge values from the others and the "Precond" port should support the composite values.

6.3.4 Program Generator from RPN

A formal specification of the target software system may be more useful if there exists a method to generate the program source codes from the verified formal

specifications. The expected source codes will be used as the prototype and any modification can be done repeatedly before the implementation of the software system begins. We have already investigated further steps towards the automatic generation of program codes from Z specification. In [49], we experimentally generate Java data object class from Z schemas.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

REFERENCES

[1] Huth, M. R. A., and Ryan M. D. Logic in Computer Science: Modelling and reasoning about systems, Cambridge University Press, 2000.

[2] Yau, S. S., and Liu, C. S. An Approach to Software Requirement Specification. In Proceedings of Computer Software and Applications Conference (COMPSAC 88) Twelfth International (1988): 83–88.

[3] Tsai, J. J. P., Weigert, T., and Aoyama, M. A Declarative Approach to Software Requirement Specification Languages. In Proceedings of International Computer Languages (1988): 414-421.

[4] Jarvinen, H. M., and Suonio, R. K. DisCo Specification Language: Marriage of Actions and Objects. In Proceedings of Conference on 11th International Distributed Computing Systems (1991): 142-151.

[5] Cheng, B. H. C., and Gannod, G. C. Abstraction of Formal Specifications from Program Code. In Proceedings of IEEE International Conference on Tools for AI (1991)

[6] Jin, L., and Zhu, H. Automatic Generation of Formal Specification from Requirement Definition. In Proceedings of First IEEE Format Engineering Methods (1997): 243-251.

[7] von der Beeck, M., Margaria, T., and Steffen, B. A Formal Requirements Engineering Method for Specification, Synthesis, and Verification. In Eighth Conference on Software Engineering Environments (1997): 131-144.

[8] Harry, A. Formal Methods Fact File: VDM and Z,Wiley & Sons, 1996.

[9] Jategaonkar, L. Transferring Formal Methods Technology to Industry. In Proceeding of the 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques (October 1998): 128-133.

[10] Hinchey, M. G. and Jarvis, S. A. Concurrent Systems: Formal Development in CSP, McGraw-Hill, 1995.

[11] Williams, A. Formal Methods Background Information [Online], Available from: http://www.cs.man.ac.uk/fmethods/background.html [2002, April 9]

[12] Bowen, J., and Hinchey, M. G. Ten Commandments of Formal Methods. IEEE Computer (1995).

[13] Larsen, P. G., Fitzgerald, J. and Brookes, T. Applying Formal Specification in Industry. IEEE Software (May 1996): 48-56.

[14] Hall, A. What does industry need from formal specification techniques. In Proceeding of the 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques (October 1998): 2-9.

[15] Alexandar, P. Best of both worlds: combining formal and semi-formal methods in software engineering. IEEE Potentials Journal (December 1995-January 1996).

[16] Spivey, J. M. An Introduction to Z and Formal Specifications. Software Engineering Journal (January 1989).

[17] Crossley, J. N., Ash, C. J., Brickhill, C. J., Stillwell, J. C., and Williams, N. H. What is mathematical logic?, Oxford University Press, 1979.

[18] Wieringa, R. A Survey of Structured and Object-Oriented Software Specification Methods and Techniques. ACM Computing Surveys 30, No. 4 (December 1998).

[19] Pressman, R. S. Software Engineering – A practitioner's approach (fourth edition), McGraw-Hill, 1997.

[20] Marca, D., and Gowan, C. SADT: Structured Analysis and Design Technique, New York: McGraw-Hill, 1988.

[21] Martin, J. Information Engineering, Book I: Introduction, New Jersey: Prentice-Hall, 1989.

[22] Coad, P., and Yourdon, E. Object-Oriented Analysis, New Jersey: Yourdon Press/Prentice-Hall, 1990.

[23] Booch, G. Object-Oriented Design with Applications, California: Benjamin/Cummings, 1991.

[24] Loomis, M., Shah, A., and Rumbaugh, J. An object modeling technique for conceptual design. European Conference on Object-Oriented Programming, LNCS 276, Springer, 1987.

[25] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. Object-Oriented Modeling and Design, New Jersey: Prentice-Hall, 1991.

[26] Booch, G., Rumbaugh, J., and Jacobson, I. The unified modeling language user guide, Addison-Wesley, 1999.

[27] Nicholl, R. A. Unreachable States in Model-Oriented Specifications. IEEE Transactions on Software Engineering 164 (April 1990) 472 – 477.

[28] Zhang, L. Combining Formal Specification Methods and Informal Specification Methods for Requirement Analysis. In the Proceeding of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing 1 (1997): 444 – 447.

[29] Jones, C. B. Software Development: A Rigorous Approach, New Jersey: Prentice-Hall International Series in Computer Science, 1980.

[30] ORA CANADA. ORA Canada: Z/EVES [Online], Available from: http://www.ora.on.ca/z-eves/welcome.html [2002, April 9]

[31] Abrial, J. R. The B-Book, Assigning programs to meaning, Cambridge University Press, 1996.

[32] Habrias, H., and Briech, B. Formal Specification of Dynamic Constraints with the B Method. In Proceeding of the 1st International Conference on Formal Engineering Methods (ICFEM '97) (November 1997): 304- 314.

[33] B-Core UK (Limited). A Comparison of Z and VDM with B/AMN [Online], Available from: http://www.b-core.com/ZVdmB.html [2002, April 9]

[34] Disco Group, The Disco Homepage [Online], Available from: http://www.cs.tut.fi/General.html [2002, April 9]

[35] Diaconescu, R., Futatsugi, K., and Iida, S. Component-based Algebraic Specification and Verification in CafeOBJ. In Proceeding of World Congress on Formal Methods in the Development of Computing Systems (FM'99) (1999).

[36] Diaconescu, R., and Futatsugi, K. CafeOBJ Report: The Language, Proof Techniques: and Methodologies for Object-Oriented Algebraic Specification, vol. 6 of AMAST Series in Computing World Scientific, 1998.

[37] Futatsugi, K., and Nakagawa, A. An overview of CAFE specification environment-an algebraic approach for creating, verifying, and maintaining formal specifications over networks. In the Proceeding of

First IEEE International Conference on Format Engineering Methods. (1997): 170 –181.

[38] Teichroew, D. Scientific and Business Applications. Communications of the ACM 8, No.1 (January 1965).

[39] Vatanawood, W., Sriratanalai, V., and Rivepiboon, W. Formal Specification Synthesis for Database Applications. In Proceedings of the International Conference on Intelligent Technologies 2000 (InTech 2000) (December 2000).

[40] Achour, C. B. Writing and Correcting Textual Scenarios for System Design. In Proceedings of 9th International Workshop on Database and Expert Systems Applications (1998): 166 –170.

[41] Kaindl, H., Kramer, S., and Kacsich, R. A Case Study of Decomposing Functional Requirements Using Scenarios. In Proceedings of the 1998 International Conference on Requirements Engineering (ICRE'98) (1998).

[42] Vatanawood, W., and Chongstitvatana, P. A Genetic Algorithm Approach to Software Components Search using Software Functional Requirement Checklist. In Proceedings of 3rd Annual National Symposium on Computational Science and Engineering (ANSCSE) (March 1999).

[43] Pooch, U. W. Translation of Decision Tables. Computing Surveys 6, No.2 (June 1974).

[44] Barden, R., Stepney, S., and Cooper, D. Z in Practice, Prentice-Hall, 1994.

[45] Vatanawood, W., and Rivepiboon, W. Formal Specification Scheme for Database Applications using Requirements Particle Networks. International Journal for Computer-Aided Engineering and Software 19, No. 8 (2002): 932-952.

[46] Polack, F. Integrating Formal Notations and Systems Analysis: using Entity Relationship Diagrams. Software Engineering Journal (September 1992): 363 - 371.

[47] Babin, G., Lustman, F., and Shoval, P. Specification and Design of Transactions in Information Systems: A Formal Approach. IEEE Transactions on Software Engineering 17, No. 8 (August 1991): 814-829.

[48] Kouno, S., Chang, H., and Araki, K. Consistency Checking between Data and Process Diagrams based on Formal Methods. In Proceedings of 20th International Conference on Computer Software and Applications (1996): 261-269.

[49] Vatanawood, W., Hirunpiwong, W., and Rivepiboon, W. n Automatic Generator of Java Data Object Class from Z Specification. In Proceedings of the 2001 International Conference on Information Technology (June 2001): 249-256.

[50] Spivey, J. M. An introduction to Z and formal specification. Software Engineering Journal 4, No.1 (January 1989)

[51] Jackson, D. Structuring Z Specifications with Views. In Technical Report CMU-CS-94-126 (1994).

[52] Vatanawood, W., Kamchornwate, W., and Rivepiboon, W. Requirements Particle Networks: An Approach to Software Functional Requirement Modelling. In Proceeding of International Conference on Applied Simulations and Modelling (ASM2000) (2000).

[53] ORA CANADA. Z/EVES Manuals [Online], Available from: http://www.ora.on.ca/z-eves/welcome.html [2002, April 9]

[54] Grimm, K. Industrial Requirements for the Efficient Development of Reliable Embedded Systems. In Proceeding of 11th International Conference of Z Users (ZUM'98) (1998): 1-4.

[55] King, S., Hammond, J., Chapman, R., and Prior, A. The Value of Verification: Positive Experience of Industrial Proof. In Proceeding of World Congress on Formal Methods in the Development of Computing Systems (FM'99) (1999): 1527-1545.

[56] Derrick, J., and Boiten, E. Refinement in Z and Object Z – Foundations and Advanced Applications, Springer Verlag, 2001.

APPENDICES

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

APPENDIX A: FINAL SPECIFICATION IN CASE STUDIES

A.1 The Complete Z Specifications for Video Shop System

[VIDEOTITLE]

VIDEOSTOCK = = Π VIDEOTITLE

[HIRER]

CLUBMEMBER = = Π HIRER

[HIRERVIDEO]

HIRINGRECORD = = Π HIRERVIDEO

Rule1Search / [VideoTitle? : VIDEOTITLE;
VideoStock? : VIDEOSTOCK;
Rule1SearchAck! : Boolean;
Rule1SearchNack! : Boolean;
Rule1SearchOut! : VIDEOTITLE |
Rule1SearchOut! = VideoTitle? ∧
Rule1SearchAck! = VideoTitle? ∈ VideoStock? ∧
Rule1SearchNack! = VideoTitle? ∉ VideoStock?]

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

Rule1Store / [VideoTitle? : VIDEOTITLE;
VideoStock?, VideoStock' : VIDEOSTOCK;
Rule1Store Ack! : Boolean;
Rule1Store Nack! : Boolean;
Rule1Store Out! : VIDEOTITLE;
Rule1Search |
VideoTitle? ∉ VideoStock? ∧
VideoStock' = VideoStock? ∪ {VideoTitle?} ∧
Rule1Store Out! = VideoTitle? ∧

Rule1Store Ack! = VideoTitle? $\in$ *VideoStock'* $\wedge$

Rule1Store Nack! = VideoTitle? $\notin$ *VideoStock']*

Rule2Search1 / *[Hirer? : HIRER;*

ClubMember? : CLUBMEMBER;

Rule2Search1Ack! : Boolean;

Rule2Search1Nack! : Boolean;

Rule2Search1Out! : HIRER |

Rule2Search1Out! = Hirer? $\wedge$

Rule2Search1Ack! = Hirer? $\in$ *ClubMember?* $\wedge$

Rule2Search1Nack! = Hirer? $\notin$ *ClubMember?]*

Rule2Search2 / *[VideoTitle? : VIDEOTITLE;*

VideoStock? : VIDEOSTOCK;

Rule2Search2Ack! : Boolean;

Rule2Search2Nack! : Boolean;

Rule2Search2Out! : VIDEOTITLE;

Rule2Search1 |

Hirer? $\in$ *ClubMember?* $\wedge$

Rule2Search2Out! = VideoTitle? $\wedge$

Rule2Search2Ack! = VideoTitle? $\in$ *VideoStock?* $\wedge$

Rule2Search2Nack! = VideoTitle? $\notin$ *VideoStock?]*

Rule2Store / *[HirerVideo? : HIRERVIDEO;*

HiringRecord?, HiringRecord' : HIRINGRECORD;

Rule2StoreAck! : Boolean;

Rule2StoreNack! : Boolean;

Rule2StoreOut! : HIRERVIDEO;

Rule2Search2 |

VideoTitle? $\in$ *VideoStock?* $\wedge$

HiringRecord' = HiringRecord? $\cup$ *{HirerVideo?}* $\wedge$

Rule2Store Out! = HirerVideo? $\wedge$

Rule2Store Ack! = HirerVideo? ∈ HiringRecord' ∧

Rule2Store Nack! = HirerVideo? ∉ HiringRecord']

Rule3Search / [Hirer? : HIRER;

ClubMember? : CLUBMEMBER;

Rule3SearchAck! : Boolean;

Rule3SearchNack! : Boolean;

Rule3SearchOut! : HIRER |

Rule3SearchOut! = Hirer? ∧

Rule3SearchAck! = Hirer? ∈ ClubMember? ∧

Rule3SearchNack! = Hirer? ∉ ClubMember?]

Rule3Store / [Hirer? : HIRER;

ClubMember?, ClubMember' : CLUBMEMBER;

Rule3StoreAck! : Boolean;

Rule3StoreNack! : Boolean;

Rule3StoreOut! : HIRER;

Rule3Search |

Hirer? ∉ ClubMember? ∧

ClubMember' = ClubMember? ∪ {Hirer?} ∧

Rule3StoreOut! = Hirer? ∧

Rule3StoreAck! = Hirer? ∈ ClubMember' ∧

Rule3StoreNack! = Hirer? ∉ ClubMember']

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

Rule4Search / [HirerVideo? : HIRERVIDEO;

HiringRecord? : HIRINGRECORD;

Rule4SearchAck! : Boolean;

Rule4SearchNack! : Boolean;

Rule4SearchOut! : HIRER |

Rule4SearchOut! = HirerVideo? ∧

Rule4SearchAck! = HirerVideo? $\in$ *HiringRecord?* $\wedge$

Rule4SearchNack! = HirerVideo? $\notin$ *HiringRecord?]*

Rule4Remove / *[HirerVideo? : HIRERVIDEO;*

HiringRecord?, HiringRecord' : HIRINGRECORD;

Rule4RemoveAck! : Boolean;

Rule4RemoveNack! : Boolean;

Rule4RemoveOut! : HIRERVIDEO;

Rule4Search |

HirerVideo? $\in$ *HiringRecord?* $\wedge$

HiringRecord' = HiringRecord? \ {HirerVideo?} $\wedge$

Rule4RemoveOut! = HirerVideo? $\wedge$

Rule4RemoveAck! = HirerVideo? $\notin$ *HiringRecord'* $\wedge$

Rule4RemoveNack! = HirerVideo? $\in$ *HiringRecord']*

Rule1 / *Rule1Search* $\wedge$ *Rule1Store*

Rule2 / *Rule2Search1* $\wedge$ *(Rule2Search2* $\wedge$ *Rule2Store)*

Rule3 / *Rule3Search* $\wedge$ *Rule3Store*

Rule4 / *Rule4Search* $\wedge$ *Rule4Remove*

VDOSHOPSPEC / *Rule1* $\vee$ *Rule2* $\vee$ *Rule3* $\vee$ *Rule4*

The Z specification is tested using Z/EVES's *try* and *prove by reduce* commands. For example, we set the goal using *try Rule1[VideoTitle? := Starwar, VideoStock? := \{ Barney \}];* and expect the result from Z/EVES to show that the new video title named 'Starwar' is in the video stock set. We then repeat the test for each rules.

A.2 The Sample Z Specifications for Van Hire System

We present a part of Z schemas are generated using the transformation rules from [39]. The transformation rules actually maintain the integrity of structural constraints among the entities and relations such as primary key, foreign key, cardinality of the relations etc. The primitive operations such as inserting, deleting, updating, are given below along with several essential state schemas indicating the data sets or entities in database of van hire system.

BOOLEAN ::= TRUE | FALSE

[DATE, FUELTYPE, STRING]

Vanclass / *[ClassName : STRING;*
FuelType : FUELTYPE]

Van / *[Reg : STRING;*
Model : STRING;
Class : STRING]

Customer / *[CustId : STRING;*
CustName : STRING;
CustAddr : STRING]

Booking / *[BookId : STRING;*
StartDate : DATE;
EndDate : DATE;
CustId : STRING;
Class : STRING;
VanNo : STRING]

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

VanclassExt / *[VanclassSet : Φ Vanclass |*
A Vanclass1, Vanclass2 : Vanclass |
Vanclass1 ∈ VanclassSet ∧
Vanclass2 ∈ VanclassSet ∧
Vanclass1 ⎣ Vanclass2 •
Vanclass1.ClassName ⎣ Vanclass2.ClassName]

VanExt / *[VanSet : Φ Van |*
A Van1, Van2 : Van |
Van1 ∈ VanSet ∧
Van2 ∈ VanSet ∧
Van1 ⎣ Van2 •
Van1.Reg ⎣ Van2.Reg ∧
A VanVar : Van •
(∃ VanclassVar : Vanclass •
VanVar.Class = VanclassVar.ClassName)]

CustomerExt / *[CustomerSet : Φ Customer |*
A Customer1, Customer2 : Customer |
Customer1 ∈ CustomerSet ∧
Customer2 ∈ CustomerSet ∧
Customer1 ⎣ Customer2 •
Customer1.CustId ⎣ Customer2.CustId]

BookingExt / *[BookingSet : Φ Booking |*
A Booking1, Booking2 : Booking |
Booking1 ∈ BookingSet ∧
Booking2 ∈ BookingSet ∧
Booking1 ⎣ Booking2 •
Booking1.BookId ⎣ Booking2.BookId ∧
A BookingVar : Booking •
(∃ VanVar : Van •

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

BookingVar.VanNo = VanVar.Reg)
A BookingVar : Booking •
(∃ VanclassVar : Vanclass •
BookingVar.VanNo = VanclassVar.ClassName)
A BookingVar : Booking •
(∃ CustomerVar : Customer •
BookingVar.CustId = CustomerVar.CustId)]

InsertBooking / *[Δ BookingExt;*
NewValue? : Booking
InsertCustomer
InsertVanclass
InsertVan |
BookingSet' = BookingSet ∪ {NewValue?}]

DeleteBooking / *[Δ BookingExt;*
dBooking? : Booking |
BookingSet' = BookingSet \ {dBooking?}]

UpdateBooking / *[DeleteBooking; InsertBooking]*

InsertCustomer / *[Δ CustomerExt;*
NewValue? : Customer |
CustomerSet' = CustomerSet ∪ {NewValue?}]

DeleteCustomer / *[Δ CustomerExt;*
dCustomer? : Customer;
DeleteBooking |
CustomerSet' = CustomerSet \ {dCustomer?}]

UpdateCustomer / *[DeleteCustomer; InsertCustomer]*

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

InsertVan / *[Δ VanExt;*
NewValue? : Van;
InsertVanclass |
VanSet' = VanSet ∪{NewValue?}]

DeleteVan / *[Δ VanExt;*
dVan? : Van;
DeleteBooking |
VanSet' = VanSet \ {dVan?}]

UpdateVan / *[DeleteVan; InsertVan]*

InsertVanclass / *[Δ VanclassExt;*
NewValue? : Vanclass |
VanclassSet' = VanclassSet ∪{NewValue?}]

DeleteVanclass / *[Δ VanclassExt;*
dVanclass? : Vanclass;
DeleteBooking
DeleteVan |
VanclassSet' = VanclassSet \ {dVanclass?}]

สถาบันวิทยบริการ

UpdateVanclass / *[DeleteVanclass; InsertVanclass]*

จุฬาลงกรณ์มหาวิทยาลัย

The following is the Z specifications of composite operations concluded from the case study.

Rule1Search / *[Van? : VAN;*
VanSet? : VANSET;

Rule1SearchAck! : Boolean;
Rule1SearchNack! : Boolean;
Rule1SearchOut! : VAN |
Rule1SearchOut! = Van? ∧
Rule1SearchAck! = Van? ∈ VanSet? ∧
Rule1SearchNack! = Van? ∉ VanSet?]

Rule1InsBooking / [Rule1InsBookingAck! : Boolean;
Rule1InsBookingNack! : Boolean;
InsertBooking
Rule1Search |
Van? ∈ VanSet? ∧
Rule1InsBookingAck! = (Van? ∈ VanSet?) ∧
Rule1InsBookingNack! = !(Van? ∈ VanSet?)]

Rule2Search / [Booking? : BOOKING;
BookingSet? : BOOKINGSET;
Rule2SearchAck! : Boolean;
Rule2SearchNack! : Boolean;
Rule2SearchOut! : BOOKING |
Rule2SearchOut! = Booking? ∧
Rule2SearchAck! = Booking? ∈ BookingSet? ∧
Rule2SearchNack! = Booking? ∉ BookingSet?]

Rule2DelBooking / [Rule2DelBookingAck! : Boolean;
Rule2DelBookingNack! : Boolean;
DeleteBooking
Rule2Search |
Booking? ∈ BookingSet? ∧
Rule2DelBookingAck! = (Booking? ∈ BookingSet?)∧
Rule2DelBookingNack! = !(Booking? ∈ BookingSet?)]

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

$Rule1 \,/\, Rule1Search \wedge Rule1InsBooking$

$Rule2 \,/\, Rule2Search \wedge Rule2DelBooking$

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

APPENDIX B: PUBLICATIONS

B.1 International Conferences

1) Vatanawood, W., Kamchornwate, W., and Rivepiboon, W. Requirements Particle Networks: An Approach to Software Functional Requirement Modelling. In Proceeding of International Conference on Applied Simulations and Modelling (ASM2000) (2000).
2) Vatanawood, W., Sriratanalai, V., and Rivepiboon, W. Formal Specification Synthesis for Database Applications. In Proceedings of the International Conference on Intelligent Technologies 2000 (InTech 2000) (December 2000).

B.2 International Journals

1) Vatanawood, W., and Rivepiboon, W. Formal Specification Scheme for Database Applications using Requirements Particle Networks. International Journal for Computer-Aided Engineering and Software 19, No. 8 (2002): 932-952.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

BIOGRAPHY

Wiwat Vatanawood received a B.S. in Computer Engineering from Chulalongkorn University, Thailand in 1985 and M.S. in Computer Science from California State University, Fullerton, U.S.A. in 1989. He is currently an assistant professor of Computer Engineering at Faculty of Engineering, Chulalongkorn University. His research interests include formal specification methods, software process modeling, and computer graphics.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย