XML-BASED FORMAL SPECIFICATION COMPREHENSION

HUANG XIAO NING

(B.Sc. Fudan University, China)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2004

Acknowledgement

At the very beginning I would like to express my deepest gratitude to my supervisor Dr. Dong Jinsong. I owe my Master program and this thesis to his continuous encouragement and support, his valuable guidance and insight through this program, and his constructive criticisms and suggestions of this thesis. I especially appreciate his patience and kindness when I come across any trouble in my research. I feel really lucky to have such a nice advisor.

Next I would like to thank Professor Stan Jarzabek and Professor Chan Chee Yong, for their wonderful suggestions of my project.

I would like to thank Sun Jing, Wang Hai, Li Yuan Fang and other officemates, for their valuable discussions and comments, their help and friendship.

This study has received financial support from the National University of Singapore. The School of Computing has provided me with excellent facilities and resources. For all of this, I am very grateful.

Many thanks to my friends in Singapore. It is their friendship that makes my days in the National University of Singapore pleasant and memorable.

Finally I am in deep grateful to my parents in China. Without their love, encouragement and understanding, I cannot finish my Master research. This thesis is dedicated to them.

Contents

List of Figures

Summary

Specification comprehension is an analytical process of a specification model. During this process the specification model can be improved. Our concept of specification comprehension comes from the idea of program understanding. It aims at utilizing some techniques to display the static and dynamic properties of a specification.

In this thesis, we propose a framework of specification comprehension for Z family formal languages(Z/Object-Z/TCOZ), particularly TCOZ language. The environment of Z-family we exploit is ZML. Three techniques helping the comprehension of a Z family specification are introduced: query, visualization and animation.

The query of a Z family specification is similar to the query of a program. Through the process of query, we attempt to display some static properties of the specification, such as properties about a class, an operation, or a cross-reference between classes.

The visualization of a Z family specification is achieved by UML projection. This projection transforms the textual specification into UML diagrams, e.g. statecharts, which illustrate the relationship between the classes of this specification.

The animation of a Z family specification aims at displaying the dynamic properties of the specification. It utilizes the transformation from Z family language to Java language to achieve an animated mapping of the original Z family document. Then through this animated document, some dynamic properties of the original one can be illustrated easily and directly.

A case tool is implemented in this thesis. This case tool provides an environment to display and edit the Z family specification, implements the query and animation functions and also links the visualization function as a module of it. This tool also supports schema checking and simple logic and semantic checking.

Chapter 1

Introduction

We start this thesis with a brief introduction to formal methods and specification languages. Then we discuss the motivation of specification comprehension. Later we demonstrate summarily the three categories of specification comprehension for Z family formal languages. This chapter ends with an overview of the organization of the whole thesis.

1.1 Motivations and Objectives

Formal methods are techniques that provide mathematical groundwork for the design of more reliable software. They help reducing the errors of a system at the early stages of design. A formal specification is a specification that utilizes some formal methods to model a system accurately. Many specification languages have been proposed. For example, some state-oriented formalisms such as VDM [1], Z [39] , Object-Z [8] model systems by an underlying state which can undergo change; some process-oriented formalisms such as CSP [14], CCS [16] , LOTOS [3] model systems as processes partaking in communication; some algebraic formalisms such as ACT1 [9], OBJ [11], Larch [15] model systems by equations related by axioms(rewriting rules); some formalisms are the combination of other formalisms, such as TCOZ(Timed Communicating Object Z) [24], which is the combination of Object-Z and Timed CSP.

A specification model can be improved during the analytical process of itself. We believe *specification comprehension* may be a new research area to pursue. Our concept of specification comprehension comes from the idea of program understanding. In our viewpoint, specification comprehension is more important than program understanding. Programs are executable, which make it easy to know whether this program satisfies users' requirement. There are also many kinds of debugger tool for each programming language, which make it convenient to check the syntax or semantic errors of a program. As to a specification, it may not be necessarily executable [12]. It is not easy to perform this analytical process. For such reasons, it is very important to develop some techniques to help the user understand the specification and to provide tools for these techniques, which is what our specification comprehension attempts to do.

In this thesis we put forward a framework of specification comprehension for Z family formal languages, especially TCOZ. The environment of Z family we exploit here is ZML(Z family on the Web through XML and UML projection facilities) [34, 35]. We introduce three parts of comprehension for XML-based Z family specifications in this thesis. The first part is the query of a Z family specification. This kind of comprehension is to answer queries about a TCOZ specification and provide useful information to the user. It itself includes five types: class query, schema query, operation query, variable query and cross-reference query. The first four types of query provide information on classes, schemas, operations and variables. The last one provides information on the cross-references between classes or schemas.

In this thesis, we also introduce an implementation of TCOZ visualization utilizing UML projection proposed by Sun et al [34, 35]. Sun et al.'s UML projection build the connection between TCOZ and UML diagrams. We find that this kind of connection can be viewed as a visualization tool of our specification comprehension for Z family languages. UML is the most popular graphical notation which is easy to understand and widely accepted by the industry. By transforming TCOZ specifications into UML diagrams, the readability and interpretability of a specification can be improved. In our work, we exploit this UML projection as a tool for visualizing Z family languages and link the transformation process of UML projection as part of our case tool.

The third part we introduce to our specification comprehension is the animation from a Z family specification to a Java program. Animation is a means of performing the validation to determine whether the requirements of a specification are the right requirements and whether they are complete. Its purpose is to exhibit the dynamic properties of a specification. In this thesis we present a simple approach of animating a TCOZ specification in Java language and utilizing this animation. We attempt to illustrate the dynamic properties of the original TCOZ specification.

At the end of this thesis, we present a case tool which provides an environment where a Z family specification can be displayed and edited in both textual and graphical form. The query and animation of Z family specifications are both implemented in this case tool, and the visualization with UML statechart is also linked as a part to it. This case tool is also able to support schema checking and simple logic and semantic checking.

1.2 Organization of the Thesis

The structure of this thesis is as follows.

Chapter 2 introduces the background of the Z family specification languages and ZML environment. It also gives a review of past and current research on specification comprehension.

Chapter 3 compares specification comprehension with program understanding, a better developed research area, where our idea of specification comprehension comes from.

Chapter 4 describes our framework of specification comprehension and illustrates the three parts of Z family comprehension using a specification of *Queue* system. Chapter 5 demonstrates the case tool - an environment for Z family comprehension. Chapter 6 concludes this thesis and highlights some possible future research directions.

Chapter 2

Background and Related Work

This chapter sets the context for later chapters. We introduce the related notations and tools of Z family languages and ZML. We also present a review of research works on specification comprehension.

2.1 Z Family Specification

In this section, we will use a simple message queue system to give a brief introduction to the Z, Object-Z and TCOZ notations.

2.1.1 Z and Object-Z

A typical Z [39] specification consists of a number of state and operation schema definitions. A state schema encapsulates variable declarations and related predicates(invariants). The system state is determined by values taken by variables which are subject to restrictions imposed by state invariants. An operation schema defines the relationship between the "before" and "after" states which are corresponding to one or more state schemas. Complex schema definitions can be composed by simple ones utilizing schema calculus.

Consider the Z model of a FIFO message queue. The given basic type *MSG* presents a set of messages. The corresponding notation for this is:

$[MSG]$

This queue contains two operations *Add* and *Delete*. *Add* operation is to add elements to the queue while *Delete* operation is to delete elements from the queue. The number of the total elements in the queue cannot be more than *max* (that is, a number larger than 100). The global constant *max* can be defined using the Z axiomatic definitions as:

$$\begin{array}{|l}
max : \mathbb{N} \\
\hline
max > 100
\end{array}$$

$\mathbb{N}$ is understood as a predefined type in Z-family languages, which represents natural numbers. The state of the queue system can be specified in Z as:

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Queue}} \\ \hline
\mathit{items} : \mathrm{seq}\, \mathit{MSG} \\ \hline
\#\mathit{items} \leqslant \mathit{max} \\ \hline
\end{array}$$

The initial state of the queue can be specified by using schema inclusion (the simplest form of Z schema calculus) as:

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{QueueInit}} \\ \hline
\mathit{Queue} \\ \hline
\mathit{items} = \langle\rangle \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{QueueInit}_c} \\ \hline
\mathit{items} : \mathrm{seq}\, \mathit{MSG} \\ \hline
\#\mathit{items} \leqslant \mathit{max} \\
\mathit{items} = \langle\ \rangle \\ \hline
\end{array}$$

Note that $\mathit{QueueInit}_c$ expands the included *Queue*. $\langle\ \rangle$ represents empty sequence. The *Add* and *Delete* operation schemas can be modelled as follows:

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Add}} \\ \hline
\Delta \mathit{Queue} \\
\mathit{item}? : \mathit{MSG} \\ \hline
\#\mathit{items} \leq \mathit{max} \\
\mathit{items}' = \mathit{items} \frown \langle \mathit{item}? \rangle \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Delete}} \\ \hline
\Delta \mathit{Queue} \\
\mathit{item}! : \mathit{MSG} \\ \hline
\mathit{items} \neq \langle\ \rangle \\
\mathit{items} = \langle \mathit{item}! \rangle^\frown \mathit{items}' \\ \hline
\end{array}$$

The variable *item?* represents an input and variable *item!* represents an output. *items'* represents the value after performing the schema operation. $\Delta Queue$ denotes that the state of the schema *Queue* will be possibly changed by the operation schema. Complex operations can be constructed by using schema calculus, e.g., a new message which push out an old message, say *Penguin*, can be specified by using the sequential composition schema operator $\fatsemi$ as:

$Penguin \mathrel{\widehat=} Add \fatsemi Delete$

Which is an (atomic) operation with the effect of an *Add* followed by a *Delete*.

Other forms of schema calculus include conjunction $\wedge$, disjunction $\vee$, implication $\Rightarrow$, negation $\neg$ and pipe $\gg$, which have been discussed in many Z text books [39].

Object-Z [8] extends the Z notation by importing object-oriented concepts to Z specification language. The main advantage is the improvement of the clarity of large specifications through enhanced structuring and incremental specification. The main Object-Z structure is the class definition. A class typically includes some type and constant definitions, a state schema, an optional initial state schema and several operation schemas that define the associated operations of the state schema in the class. An Object-Z class is a template for *objects* of that class: for each such object, its states are instances of the class' state schema and its individual state transitions conform to individual operations of the class. An object is said to be an instance of a class and to evolve according to the definition of its class.

Consider the *Queue* system again for example. The specification in Object-Z is as follow:

Queue

$items : \mathrm{seq}\, MSG$

$\#items \leqslant max$

INIT

$items = \langle\ \rangle$

Add

$\Delta(items)$
$item? : MSG$

$items' = items \frown \langle item? \rangle$

Delete

$\Delta(items)$
$item! : MSG$

$items \neq \langle\ \rangle$
$items = \langle item! \rangle \frown items'$

This is a single class example with two operation schemas. The Δ list means state variables in this list may change after the operation. The *Queue* object starts with the empty set *items* and evolves by performing either *Add* or *Delete* operations. In operation *Add* an input message (defined by *item*?) is accepted by the queue provided the queue has not reached its maximum size. In the operation *Delete* the first message (defined by *item*!) leaves the queue provided that the queue is not empty and the size of the queue reduces by one after the operation.

Operations in Object-Z are atomic. An Operation may consist of several declarations and predicates. It's difficult to use the standard Object-Z semantics to model a system composed by multi-threaded component objects whose operations have duration.

Inheritance is a way of building up the specification incrementally. The functionality, modularity and reusability of Object-Z is greatly expand by incorporating inheritance, which is the major extension to pure Z.

The class *TwoQueue* defines a class that has two message queues. The operations *Join*, *Leave* and *Transfer* are defined incrementally from *Queue*.

TwoQueue

$$q1, q2 : Queue$$

$$Join \mathrel{\widehat=} q1.Add$$
$$Leave \mathrel{\widehat=} q2.Delete$$
$$Transfer \mathrel{\widehat=} q1.Delete \| q2.Add$$

2.1.2 TCOZ Features

The formal language of specification we use in this thesis is Timed Communicating Object Z (TCOZ) [24]. TCOZ is a combination of event-oriented Timed CSP [29] and state-oriented Object-Z [8]. TCOZ has both the advantages of Object-Z in modelling complex data and state and the advantages of Timed CSP in modelling real-time concurrency. Besides the primary specification structure of Object-Z, TCOZ also adopts the channel based communication mechanism and the sensor/actuator mechanism of CSP. With such advantages TCOZ is a promising candidate for complex systems design.

In this section we briefly consider various aspects of TCOZ. A detailed introduction to TCOZ and its Timed CSP and Object-Z features may be found elsewhere [23].The formal semantics of TCOZ is also documented [25].

1. A model of time

In TCOZ, *seconds*, the SI standard unit of time [17], is used to represent all timing information. Hayes and mahony [13] have extended the Z typing system to support the use of standard units of measurement. Thus, time quantities are represented by the type

$\mathbb{T} == \mathbb{R} \odot \mathbb{T}$,

where $\mathbb{R}$ represents the real numbers and T is the SI symbol for dimensions of time.

2. Interface - channels, sensors and actuators

   In TCOZ channels play a role as communication interfaces between objects. TCOZ allows the declaration of channels in state schema. If c is a communication channel, it must be declared in the state schema to be of type **chan**. Channels may carry communications of any type and are viewed as shared rather than as encapsulated entities. To complete the synchronizing CSP channel mechanism, sensors and actuators are also adopted in TCOZ as a non-synchronizing shared mechanism. The declaration of $s : X$ ***sensor*** provides a channel-like interface to input a shared variable s. The declaration of $s : X$ ***actuator*** provides a local-variable-like interface to output a shared variable s. TCOZ with sensor and actuator can be a good candidate for specifying open control systems. Mahony and Dong [26] presented detailed discussion on TCOZ sensors and actuators.

3. Active objects

   Active objects are objects that have their own threads of control, while passive objects are controlled by other objects in a system. In TCOZ, the behavior of active objects of a given class is represented by an identifier MAIN (which indicates a non-terminating process and is optional in a class definition). The objects of a class are active objects if MAIN operation appears in this class definition. Class defined for passive objects will not have MAIN definition but may contain CSP process constructors. if ob_1 and ob_2 are active objects of the class C, then the independent parallel composition behavior of the two objects can be represented as ob_1 ||| ob_2, which means ob_1.MAIN ||| ob_2.MAIN.

4. Semantics of TCOZ

   The blended state/event process model which forms the basis for the TCOZ semantics is detailed in a separate paper [25]. TCOZ interprets Z operations as processes. Operation schemas are modelled by a sequence of update events that achieve the state change. The process model of TCOZ are tuples consisting of: an *initial* state; a *trace* (a sequence of time stamped update-events), a *refusal* (a record what and when events are refused by the process), and a *divergence* (a record of if and when the process diverged). The trace/refusal pair is called a *failure* and the overall models the state/failures/divergences model. At any given time, the state of the process is the initial state updated by all of the updates that have occurred up to that time. If an event trace terminates (that is if a termination event $\surd$ occurs), then the state at the time of terminations is called the *final* state. All initial states and update traces (terminated with a $\surd$) compose the process model of an operation schema. If no legal final state exists for a given state, the operation diverges immediately.

5. Network topology

   TCOZ adopts a graph-based approach to represent the network topology [27]. For example, consider that processes A and B communicate privately through the interface *ab*, processes A and C communicate privately through the interface *ac*, and processed B and C communicate privately through the interface *bc*. The network topology of A,B and C may be described by

   $\| (A \xleftrightarrow{ab} B;\ B \xleftrightarrow{bc} C;\ C \xleftrightarrow{ca} A)$.

   Other forms of usage allow network connections with common nodes to be run together, for example

   $\| (A \xleftrightarrow{ab} B \xleftrightarrow{bc} C \xleftrightarrow{ca} A)$,

   and multiple channels above the arrow, for example, if process D and F communicate privately through the channel/sensor-actuator df_1 and df_2, then

   $\| (D \xleftrightarrow{df_1, df_2} F)$.

   The above approach implies that the basic structure of a TCOZ document is the same as for Object-Z. TCOZ varies from Object-Z in the structrue of class definitions which may include CSP channel and process definitions. For instance, an active Queue can be derived from the previous (Object-Z) *Queue* model as:

ActiveQueue

Queue

$t_j, t_l : \mathbb{T}$
$in, out : \mathbf{chan}$

$Join \mathrel{\widehat{=}} [item : MSG \mid \#items < max] \bullet in?item \longrightarrow Add \bullet DEADLINE t_j$
$Leave \mathrel{\widehat{=}} [items \neq \langle\,\rangle] \bullet out!head(items) \longrightarrow Delete \bullet DEADLINE t_l$
$MAIN \mathrel{\widehat{=}} \mu\, Q \bullet Join \Box Leave;\ Q$

2.2 ZML Environment for Z Family

ZML(Z family Markup Language) is an XML approach to define a customized markup language for the Z family notations. It creates a standard environment for constructing formal specifications on the web in XML rather than in LaTeX. In addition, through ZML, web technology can easily be used in Z-family based software design and development.

The process and techniques for ZML which are utilized in this thesis are illustrated in figure 2.1. Firstly, an XML Schema is used to define the ZML structure syntax for Z family languages. Then through this XML schema an XML document which defines a Z family specification is produced. This XML document then is parsed by an XML parser and processed by an XSL processor and finally can be displayed on the web. More about the ZML environment can be found in [36]. In this thesis, the ZML format document of Z family specification is the object of our specification comprehension.

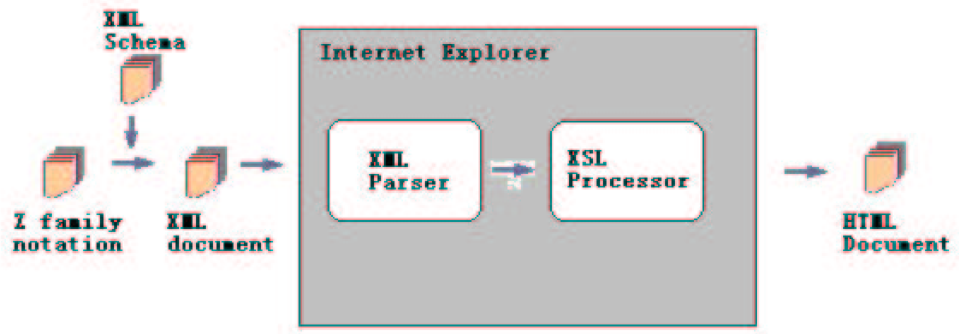


Figure 2.1: ZML process

2.3 Related Works

In this section we introduce the recent works on specification comprehension and Sun et at's work of UML projection from TCOZ to statechart.

2.3.1 Recent Works on Specification Comprehension

Recent research of specification comprehension has focused mostly on the comprehensibility (how much a user understand a specification under given conditions) of a formal language of specification, which elements affect it and how to utilize it to make a comparison among methodologies.

For example, to investigate whether formal specifications are more difficult to read than code, Snook and Harrison made an experiment [31] between 36 subjects who had been taught a course on formal methods and a similar length course on the

Java programming language. A short specification written in Z and a corresponding program implemented in Java were allocated at random to these subjects. Later the subjects were given a questionnaire to test their comprehension of the materials they had been given. At the end of this experiment they drew a conclusion that a Z specification was no more difficult to read than Java.

Another experiment trying to assess whether comprehensibility is affected by the structure in a formal Z specification was conducted by Finney, Fenton and Fedorec [10]. They provided three specifications written in Z varied in lengh and complexity. Subjects were allocated to these three specifications and then were given a questionnaire to find how much they understood their allocated specification. The results showed that the comprehensibility is neither improved by the modularization of a Z specification nor by reducing the size of the modules.

Utilizing specification comprehension to make a comparison among methodologies is also very common. For instance, two methodologies - FOOM(Functional and Object Oriented Methodology) and OPM(Object-Process Methodology), were compared from the point of view of user comprehension of specifications in [19]. And OMT(Object Modelling Technique) was also compared with OPM to discuss the model multiplicity problem by experimenting with real-time specification methods in [28].

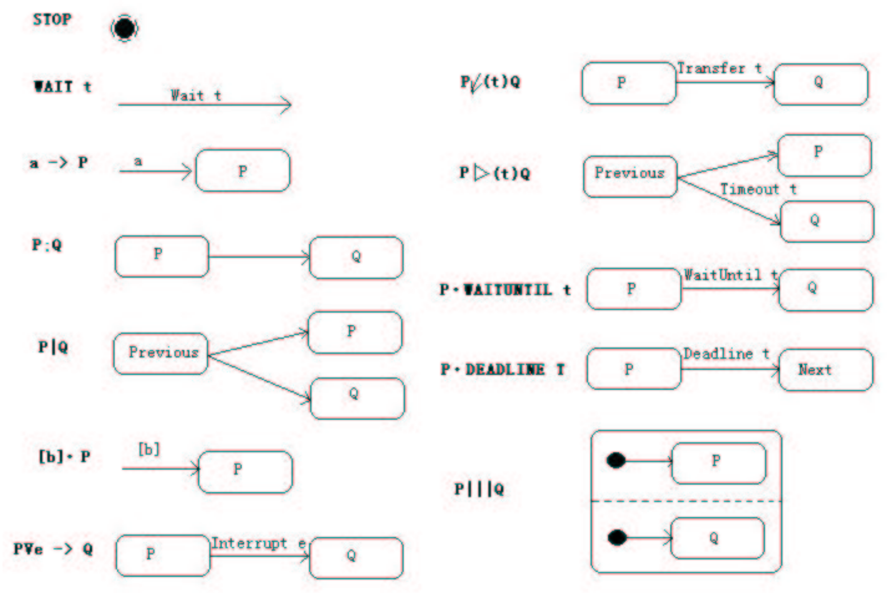


Figure 2.2: UML projection rules, excerpted from [6]

2.3.2 UML Projection by Statechart

In this section we introduce Sun et al.'s work on UML projection. This work builds the connection between TCOZ language and UML diagrams, such as statecharts. We find that this connection can be viewed as a powerful tool in terms of visualization in our work of specification comprehension for Z family languages.

UML projection transforms TCOZ specifications to UML diagrams, such as statecharts. These statecharts are easier to read and understand by users.

The key ideas of the projection are:

- UML is extended with TCOZ communication interface types - **chan**, **sensor** and **actuator**.

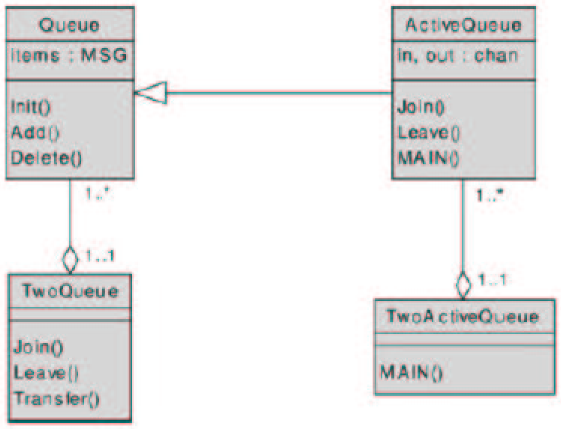


Figure 2.3: UML class diagram for *Queue* system, excerpted from [34]

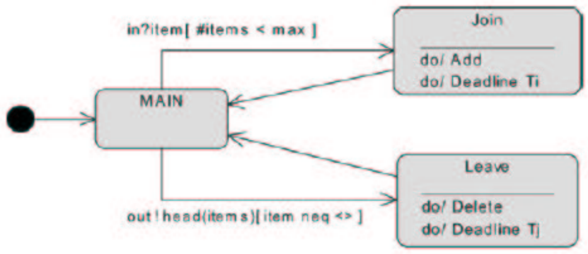


Figure 2.4: *ActiveQueue* statechart diagram, excerpted from [34]

- States of the UML statechart diagrams are identified with the TCOZ processes(operations) and the state transition links are identified with TCOZ events/guards.

Thus UML diagrams can be seen as the visual projections from a unified formal TCOZ model. Figure 2.2(excerpted from [6]) shows the detailed transformation rules from TCOZ behaviour models to UML statecharts.

In figure 2.3(excerpted from [34]), the UML class diagram depicts the *static* view of the four graph classes of the *Queue* system. This diagram was generated automatically. All attributes and operations match their definitions in the TCOZ model.

A *dynamic* view of the class *ActiveQueue* can be depicted by the statechart diagram in figure 2.4(excerpted from [34]).

Chapter 3

From Program Understanding to Specification Comprehension

This chapter describes briefly some recent research works in program understanding. Then a comparison between program understanding and our formal specification comprehension for Z family in this thesis is discussed.

3.1 The Research of Program Understanding

Program Understanding or Code Cognition is a central activity during software maintenance, evolution, and reuse [38]. It is recognized widely as a significant activity in software development and maintenance. Many important research works have been done during recent decades. In this section we introduce briefly some important works in this area. More details can be found in [4, 30, 20, 21, 22, 33, 32, 2].

1. Brooks model

   Brooks [4] argues in his theory that the process of program comprehension is finished when a complete set of mappings from the top level domain(that is, problem domain) to the bottom level domain(that is, program domain) can be made . It is the developer of the software who initially produces these mappings, whereas it is the maintainer who must recover them. This model is constructed in a top-down and breadth-first manner.

2. Shneiderman and Mayer model

   Shneiderman and Mayer [30] propose that program understanding is based upon three main type of knowledge: *syntactic knowledge*, *general semantic knowledge*, and *task related semantic knowledge*. *Syntactic knowledge* includes syntactic details of the programming language, such as keywords, language syntax , library routines and even hardware specific details. *General semantic knowledge* is composed of high level concepts such as tree traversal constructs or low level concepts such as the FOR...NEXT loop. *Task related semantic knowledge* relates directly to the problem domain, for example the semantic meaning of various program portions.

3. Letovsky model

   Letovsky's work [20] is an empirical study of programmers who attempt to maintain unfamiliar code. He proposed a cognitive model which was subdivided into three components: a knowledge base, a mental model and an assimilation process. He also suggests the assimilation process may occur in a bottom-up manner or in a top-down manner, where the bottom level is the source code and the top level is the most abstract view of the program.

4. Littman et al. model

   Littman et al. [22] propose two distinct strategies that program comprehension is based on: the *systematic* and the $as-need$. The *systematic* includes extensive studying of the static and dynamic properties of a program to acquire the program structure and causal interactions among program components. The $as-need$ helps the maintainer concentrate on the program areas which are likely to require modification.

5. Soloway, Adelson, Ehrlich and Letovsky

   Soloway and Ehrlich [33] think that programs are composed from programming plans which are used to meet the needs of the specific problem. In later works Letovsky and Soloway [21] identify the fact that the time constraints require the maintainer concentrate on localized areas of the code which they believe are required to maintain. And these plans are described as delocalized plans. Soloway, Adelson and Ehrlich [32] later propose a top-down model based on the notion of programming plans.

Based on these research works, a framework is developed [37] to provide a means to classify different approaches to program understanding. This framework is developed in three steps:

1. Investigate the cognitive aspects of program understanding, such as problem factors and cognitive models.

2. Identify some canonical activities of program understanding: data gathering, knowledge organization, and information exploration.

3. Categorize the program understanding tools and techniques along several dimensions. Categories include domain applicability, task support, and toolset extensibility.

A set of tools supporting program understanding are also developed, such as Static Program Analyzers(SPAs). SPAs are interactive tools that enhance program understanding during maintenance by answering queries about programs. It must process different source programs and answer different types of program queries. For example, Jarzabek [18] presents a design of flexible, source language-independent SPA with *PQL*, a program query language which is a conceptual level notation to specify program queries and program views.

3.2 From Program Understanding to Specification Comprehension

We attempt to apply the ideas of program understanding demonstrated in last section to our specification comprehension, utilizing the special properties and ZML environment of Z family notations.

3.2.1 What Affect Specification Comprehension?

The problem of understanding programs depends on multiple factors [37], such as the comprehensibility of programs, the experience and creativity of software maintainers, and the sheer size and complexity of programs. Similarly, the problem of understanding a specification also depends on many factors.

Firstly, specification languages differ significantly from one another in terms of their comprehensibility. For example, process-oriented specification languages may be harder to comprehend than state-oriented specification languages, and algebraic specification languages the hardest.

Secondly, the knowledge base and experience of a user affects directly the comprehension. For instance, a user who is familiar with object-oriented concept will feel Object-Z not difficult to comprehend for Object-Z is a specification language introducing some object-oriented properties to Z. However this user may suffer greatly from reading a CSP specification.

The last but most important factor that affects specification comprehension is the complexity of a specification itself. Unstructured, poor-design, redundant or just a large-size specification may be arduous to understand for a user.

3.2.2 The Descriptive Model

A significant step of program understanding is how to identify the components and their interrelationships of a program and then create representations of the program in a more recognizable form or at a higher abstraction level. This step facilitates the understanding process through the identification of its components and the discovery of their relationships.

With the same purpose, in our specification comprehension we also need to build a model to represent Z family specifications. The environment of ZML here is of great help. As we have described in last chapter, after formalizing Z family(Z/Object-Z/TCOZ) syntax in a formal model, we build a ZML environment using XML Schema. This provides a bridge from Z family specification to web environment and then we can apply some mature web technologies to our specification comprehension. This XML Schema document describes the structure of Z family notations in XML, defines the contents of all elements, the order and cardinality of sub-elements and data types of some of the elements. And then using the XML parser(e.g. parser for DOM(Document Object Model) [5]) we change this XML document into another represented structure(e.g. DOM) to facilitate the process of comprehension.

3.2.3 Activities of Specification Comprehension for Z family

Canonical activities of program understanding include data gathering, knowledge organization and information exploration. Information exploration is the final and the most important activity. It includes navigating through the descriptive model that represents the information, analyzing and filtering this information, and using various presentation mechanisms to clarify the resultant information.

In our specification comprehension, after achieving a descriptive model for Z family notations, we can exploit this model to hunt the information we need. For example, answering the question related to the Z family specification document, illustrating the specification with graphics, demonstrating the dynamic property of TCOZ specification utilizing the animation of TCOZ. These lead to the three activities we will introduce in this thesis: *query*, *visualization* and *animation* of Z family languages. In next chapter,we will demonstrate in detail these three parts of activities of our specification comprehension.

Chapter 4

Specification Comprehension for Z Family Languages

This chapter first illustrates the framework of specification comprehension. Then the activities of *query*, *visualization* and *animation* of Z family language, especially TCOZ, are demonstrated in detail.

4.1 The Framework

In this section we present a framework of specification comprehension. This framework is implemented in three steps:

1. *Data and Format*: First of all, the data(that is, the specification) and its format are the basis of our comprehension. In this thesis, TCOZ specification in XML format(ZML) is adopted.

2. *Descriptive Structure*: The to-be-comprehended specification must be put into a structure that facilitates efficient storage and retrieval and permits analysis of information and internal relationships. In this thesis, we parse an XML document into a tree structure of *Document Object Model*(DOM) [5] recommended by W3C. This tree structure permits us to create, remove, change and traverse the ZML file conveniently.

3. *Information Exploration*: This is the final yet most important step. The exploration we put forward so far in this thesis includes query, visualization and animation, which will be elaborated on in next sections.

Figure 4.1 shows the framework of our TCOZ comprehension.

4.2 Query of TCOZ Specification

Query is an important activity of TCOZ comprehension. This process enhances the understanding of the static properties of a TCOZ specification by answering

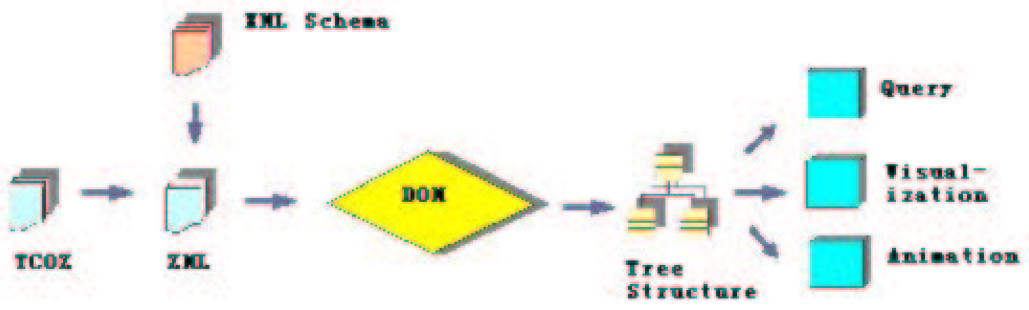


Figure 4.1: The Framework of TCOZ Comprehension

queries about the document or providing useful information on the specification. This information might be, for example, in this specification, which class contain *OP* operation; which class is the subclass of class *A*, etc.

According to which object this information is about, we divide the query of TCOZ into five types: *class query*, providing information on classes; *schema query*, providing information on schemas; *operation query*, providing information on operations; *variable query*, providing information on variables; and $cross - reference\ query$, solving queries related to the cross-references between classes or schemas.

4.2.1 Information on Classes

The information on classes can be subdivided further into information on inheritance hierarchy and details of class. In this chapter, we utilize the full *Queue* system to illustrate our TCOZ comprehension. The *Queue* system includes a basic type definition [*MSG*], an axiomatic definition *max*, a schema *Queue* and its initial state schema *QueueInit*, two operation schemas *Add* and *Delete* and four classes with inheritance relationship: *Queue*, *ActiveQueue*, *TwoQueue* and *TwoActiveQueue*.

$[MSG]$

$$\begin{array}{|l}
max : \mathbb{N} \\
\hline
max > 100
\end{array}$$

Queue

$$\begin{array}{|l}
items : \mathrm{seq}\, MSG \\
\hline
\# items \leqslant max
\end{array}$$

QueueInit

$$\begin{array}{|l}
Queue \\
\hline
items = \langle\rangle
\end{array}$$

Add

$$\begin{array}{|l}
\Delta Queue \\
item? : MSG \\
\hline
\# items \leq max \\
items' = items \frown \langle item? \rangle
\end{array}$$

Delete

$$\begin{array}{|l}
\Delta Queue \\
item! : MSG \\
\hline
items \neq \langle\ \rangle \\
items = \langle item! \rangle \frown items'
\end{array}$$

Queue

$$\begin{array}{|l}
items : \mathrm{seq}\, MSG \\
\hline
\# items \leqslant max
\end{array}$$

INIT

$$\begin{array}{|l}
items = \langle\ \rangle
\end{array}$$

Add

$$\begin{array}{|l}
\Delta(items) \\
item? : MSG \\
\hline
items' = items \frown \langle item? \rangle
\end{array}$$

Delete

$$\begin{array}{|l}
\Delta(items) \\
item! : MSG \\
\hline
items \neq \langle\ \rangle \\
items = \langle item! \rangle \frown items'
\end{array}$$

ActiveQueue

Queue

$$\begin{array}{|l}
t_j, t_l : \mathbb{T} \\
in, out : \mathbf{chan}
\end{array}$$

$$\begin{array}{l}
Join \mathrel{\widehat=} [item : MSG \mid \# items < max] \bullet in?item \longrightarrow Add \bullet DEADLINE t_j \\
Leave \mathrel{\widehat=} [items \neq \langle\ \rangle] \bullet out!head(items) \longrightarrow Delete \bullet DEADLINE t_l \\
MAIN \mathrel{\widehat=} \mu\, Q \bullet Join \Box Leave;\ Q
\end{array}$$

TwoQueue

$q1, q2 : Queue$

$Join \mathrel{\widehat=} q1.Add$
$Leave \mathrel{\widehat=} q2.Delete$
$Transfer \mathrel{\widehat=} q1.Delete \| q2.Add$

TwoActiveQueue

$q1 : ActiveQueue[talk/out]$
$q2 : ActiveQueue[talk/in]$

$MAIN \mathrel{\widehat=} \|(q1 \mid [talk] \mid q2)$

The inheritance relationship between classes is one of the usual problems that confuse the specification analyzer, especially those facing a large-size complex specification.

In *Queue* system, for instance, *ActiveQueue* is the subclass of *Queue*, while *Queue* is the superclass of *ActiveQueue*. Although *TwoQueue* and *TwoActiveQueue* also utilize the definition of *Queue*, they are not the subclass of *Queue*. They are instantiations(reference)of *Queue*, which will be further illustrated in later sections.

$$Queue \xleftarrow{inheritance} ActiveQueue$$

$$Queue \xleftarrow{reference} TwoQueue$$

$$ActiveQueue \xleftarrow{reference} TwoActiveQueue$$

Another kind of information user might be interested in is the details of a class. During the process of maintaining a large-scale specification, when questions about a class is raised, the user perhaps won't go over the whole specification, especially when the classes and their relationship is complicated. So providing some succinct information quickly to users becomes very important. For the *Queue* system, the user might require such information: where *ActiveQueue* comes from; what component of it is visible; what variables and operations it has; if *ActiveQueue* is stemmed from *Queue*, are there any other variables and operations added to it; is there any class stemmed from *ActiveQueue*, etc. When user wants to know more information about *ActiveQueue*, below is what we can provide:

Class Name: *ActiveQueue*

Subclass: *None*

Superclass: *Queue*

Formal Parameter: *None*

State Variable: T_i, T_j, *in*, *out*

Operation: *Join*, *Leave*, *MAIN*

Visibility List: *None*

The subclass is *None* means that no class inherits from *ActiveQueue*. Visibility List is *None* means that all components of this class is visible. The details of state variables and operations are hidden here.

4.2.2 Information on Schemas

Similarly users of a TCOZ specification may also encounter a schema and want to gather information of this schema without going over the whole specification. There is no inheritance hierarchy between schemas. The information we can provide is what's the use of this schema, such as where this schema is used and how it's used – just being included or being modified, etc. Take the *Queue* system for an instance, when users want to know more about the schema *Queue* (not the class *Queue*), the information we can offer here is:

Schema Name: *Queue*

Used by: *QueueInit – included*

Add – modified

Delete – modified

4.2.3 Information on Operations

When encountering operations, users might be concerned in which classes they are used and their visibility in these classes, as well as their specified functions. The visibility of an operation influences directly the usage of it. An operation invisible means that this operation can only be referenced inside the class that defines it. The users cannot reference and use this operation successfully outside this class. For example, about the operation *Leave*, information we can provide is:

Operation Name: *Leave*

In which Class: *TwoQueue – Visible*

ActiveQueue – Visible

If the user needs, the detail of this operation is also presented to make clear its specified function. However no modification is permitted:

$TwoQueue.Leave \widehat{=} q2.Delete$

$ActiveQueue.Leave \widehat{=} [\text{items} \neq \langle\ \rangle] \bullet out!head(items) \longrightarrow Delete \bullet\ DEADLINE\ t_l$

4.2.4 Information on Variables

Information on variables in a specification might be: is this variable a state variable or an input/output variable? Is this a primary or secondary variable? Where is this variable defined(in which class and further in which operation)? Is it visible? If it is an input/output variable, what is its original form? Above are the frequently asked questions about a variable as well as the specified definition of this variable. For example, we provide information of variable *in*:

a state variable

in class ActiveQueue

visible

the type of ***chan***

The information of variable *item*?:

an input variable

in operation Queue.Add

invisible

the type of MSG

original form is <u>*item*</u>

4.2.5 Information on Cross-reference

Firstly we clarify that the cross-reference here means instantiation of a class or a schema. Strictly, *inheritance* is also a kind of reference. But because we have discussed inheritance hierarchy in "Information on Classes" part, here we leave it out. *Class Union*, such as $C \mathrel{\widehat=} A \cup B$, might also be considered as a type of reference too. However, according to the definition of Class union, C here may not be a class necessarily, and we also leave it out in this thesis.

In this thesis we discuss two types of references:

a : A ,

a :↓ A

where *a* is an object and *A* is a class that *a* references. That is, *a* is an instantiation of *A*. The latter type is a kind of *Polymorphic Object Reference* because the class of this reference is not uniquely determined.

Object reference is a means to produce a concrete instance of a class. The components of a system are not the classes but the objects of them. The object can be used according to the class's interface and its behavior is consistent with that defined by the schemas of the class. Take the *Queue* system for example, class *Queue* is referenced in class *TwoQueue* to produce two instances of it: objects q_1 and q_2. They all behave as class *Queue* to build the new class *TwoQueue*. In class *TwoActiveQueue*, two objects q_1 and q_2 also reference the class *ActiveQueue*. However the output and input variables (*in* and *out*) of these two objects are renamed respectively to a unified name *talk*. This kind of renaming builds a connection between q_1 and q_2 and constructs the class *TwoActiveQueue*. Figure 4.2 shows the reference relations between *Queue*, *TwoQueue* and *TwoActiveQueue*.

4.3 Visualization of TCOZ Specification

As a combination of state-oriented Object-Z and process-oriented CSP, TCOZ is very suitable to model software systems. However, without relevant mathemati-

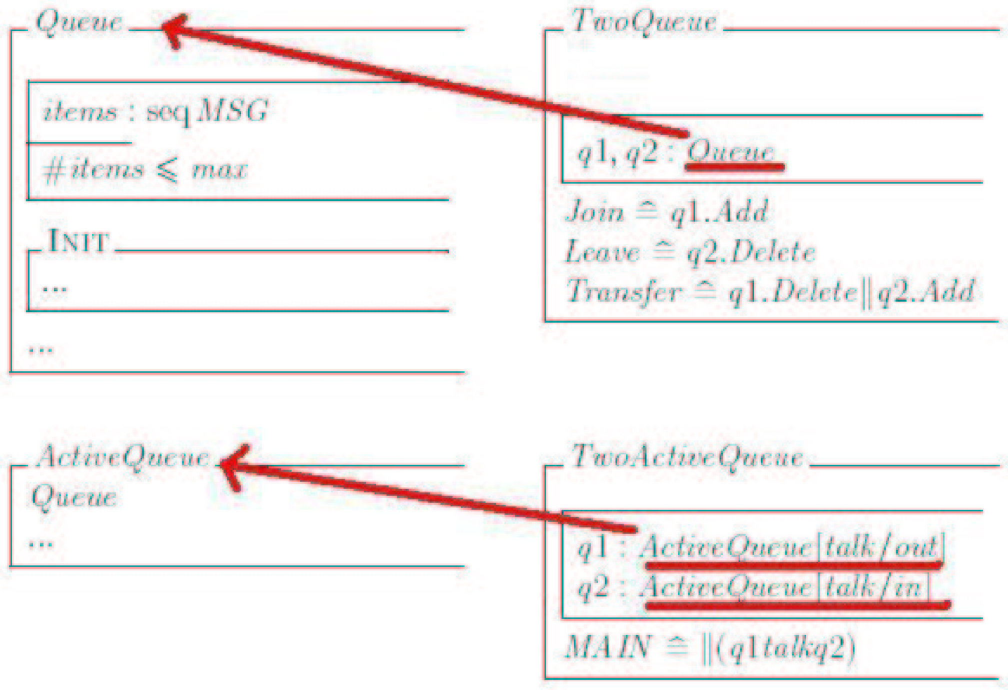


Figure 4.2: The reference between classes

cal background, TCOZ is very difficult to understand by software engineers. An useful method to interpret a TCOZ specification well is attempting to visualize the specification to UML diagrams. With these diagrams the specification can be easy to understand. This is why we view visualization as part of our formal specification comprehension. In Sun et al.'s work, a connection between TCOZ and UML diagrams has been built. We utilize this connection as a tool of our visualization, transforming a TCOZ specification to UML diagrams, such as statecharts. In "related works" section, we have introduced the work of UML projection.

4.4 Animation of TCOZ Specification

Query and visualization of a specification mainly focus on the static properties of this specification. As for the dynamic properties, we need another effective method to illustrate them. In this thesis, we utilize the animation of Z family languages to achieve this goal. The purpose of animation is to validate the requirements captured by exhibiting the dynamic properties of a specification. Animation is not a real computer system with the detailed functionalities.

Generally speaking, any programming language could be used for animation. However each programming languages has some specialized features suitable for particular types of problems. For example, Prolog is good at AI programming, PowerBuilder is good at database application and so on. The degree of similarity in syntax and semantic between formal notation and animation language should be the first criterion of selection.

Because most animation languages have differences from the formal specification notations, an equivalent library which handles all those specification constructs is indispensable. Thus the completeness of the existing library compared to the formal notation could be the second measure for the selection.

In this thesis, we choose Java programming language as our animation language for Z family notations, especially TCOZ. Java is an object-oriented, multi-thread-supported programming language with high security, robustness and running efficiency. Java is a perfect embodiment of object orientation concept and also supports multi-thread synchronization. Object orientation in Object-Z, concurrency in CSP and the combination of the two in TCOZ all could find a majority of their correspondences in Java. With the help of proper library functions, integrated notations such as TCOZ could be well animated in Java.

4.4.1 Translation Rules

The translation guideline from TCOZ to Java is the same as offering an executable semantics of TCOZ in Java. Some rules are defined as follows.

- Data types are referred to as given sets. Each data type is basically a set of possible values a variable can have.
- Sequence is referred to as *Vector* structure type in Java. Set and corresponding functions are referred to as the corresponding library methods (set library class).
- TCOZ classes is referred to as Java classes with inheritance expanded.
- Type and function definitions local to a TCOZ class is referred to as local declarations and functions in a Java class.
- The type declaration of the state schema in TCOZ class is referred to as one of the invariants in Java. The initial schema is referred to as the constructor function in the related Java class. The operation schemas are referred to as methods in the related Java class.

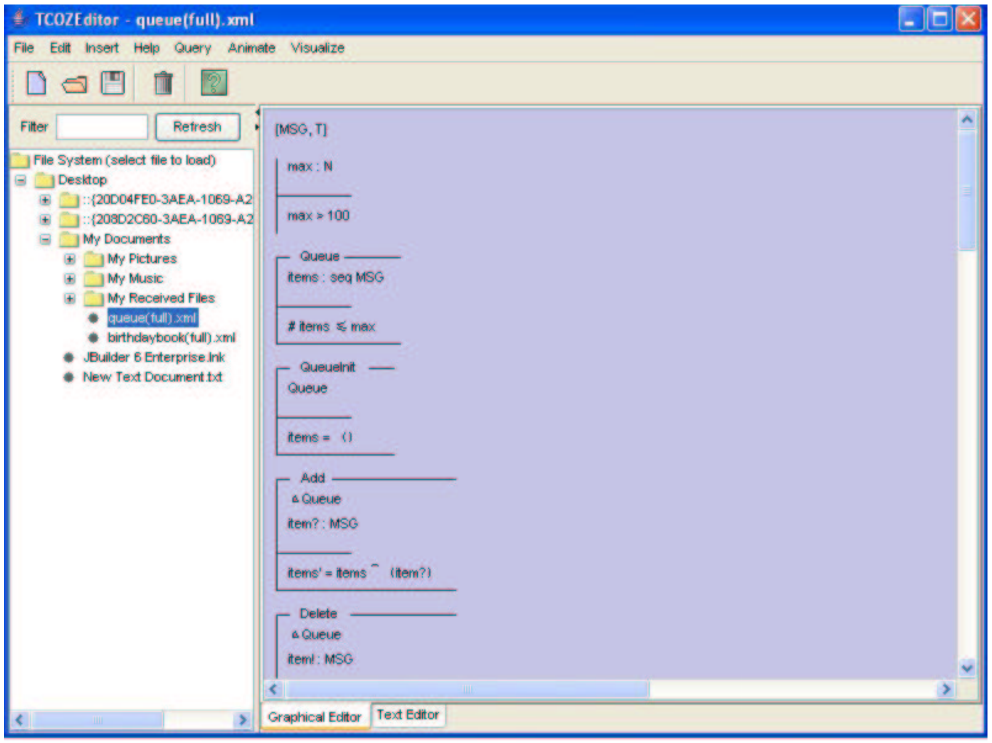
- Object reference is implemented by the instantiation (obtaining an object)of a Java class.
- Operations not in the visible list in a TCOZ class are defined as private operations in a Java class; Operations in the visible list are defined as public operations.
- Channel and sensor/actuator are referred to pre-defined Java class.

The implementation will be introduced in next chapter by a case study.

Chapter 5

A Case Tool

In this chapter we present a case tool which provides an integrated environment for specification comprehension for Z family.

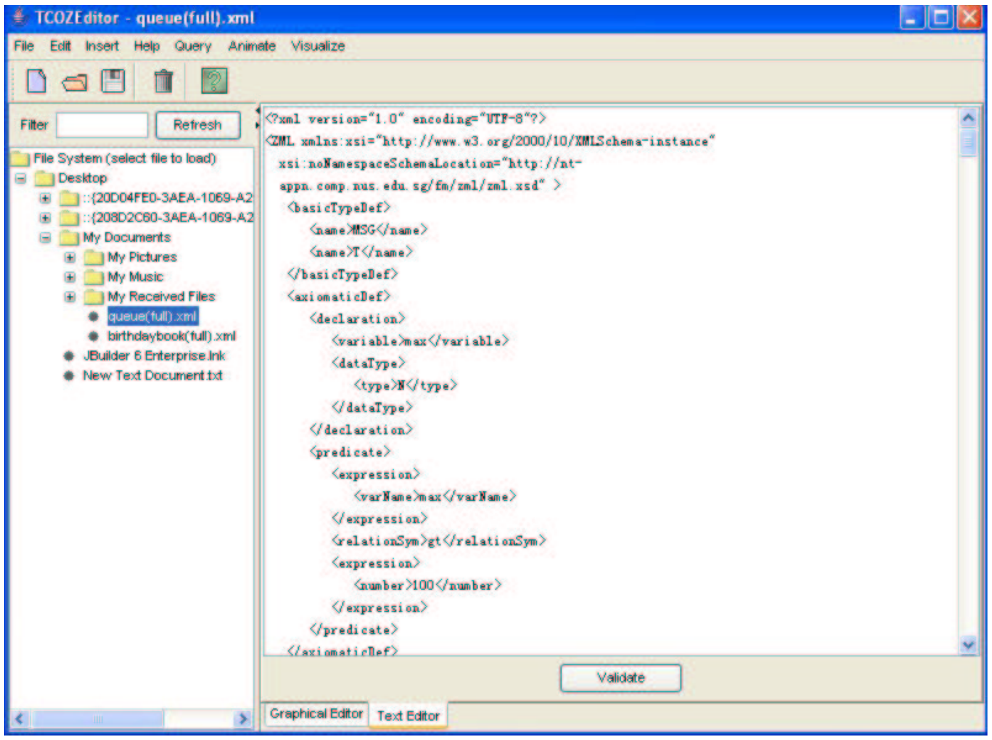


```
[MSG, T]

max : N
max > 100

Queue
items : seq MSG
# items ≤ max

QueueInit
Queue
items = ⟨⟩

Add
ΔQueue
item? : MSG
items' = items ⁀ ⟨item?⟩

Delete
ΔQueue
item! : MSG
```

Figure 5.1: The graphical displaying of *Queue* system

5.1 The Case Tool

In this section we demonstrate a case tool which provides an environment where a Z family specification can be displayed and edited in both textual and graphical form. This case tool is also able to support schema checking and simple logic and semantic checking. Figure 5.1 is the main user interface of this case tool. The right hand part is the main editing field. User can choose either Graphical Editor or Text Editor(See figure 5.2). The left part is a file system tree which makes user search and select files easier.



```
<?xml version="1.0" encoding="UTF-8"?>
<ZML xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://nt-
  appn.comp.nus.edu.sg/fm/zml/zml.xsd" >
   <basicTypeDef>
      <name>MSG</name>
      <name>T</name>
   </basicTypeDef>
   <axiomaticDef>
      <declaration>
         <variable>max</variable>
         <dataType>
            <type>N</type>
         </dataType>
      </declaration>
      <predicate>
         <expression>
            <varName>max</varName>
         </expression>
         <relationSym>gt</relationSym>
         <expression>
            <number>100</number>
         </expression>
      </predicate>
   </axiomaticDef>
```

Figure 5.2: The text displaying of *Queue* system

5.1.1 Load and Edit

The case tool allows loading and editing Z family specifications. User can open a new file. An XML parser is used at this point to check the loaded file. Only valid ZML file will be loaded. After the ZML file is loaded successfully, user can edit this file either in graphical editor or in text editor. The edit operation in the text editor is direct but not convenient. Here we explain the editor in graphical editor. If user single click at some point in the graphical editor, the editor will check whether that point belongs to any component in the menu list of Figure 5.3. If it does, the corresponding component will be highlighted. If user double click at this point, user will be allowed to edit. For an instance, let's load "queue(full).xml" to this

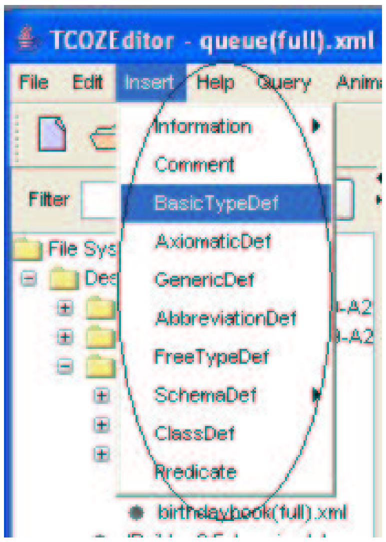
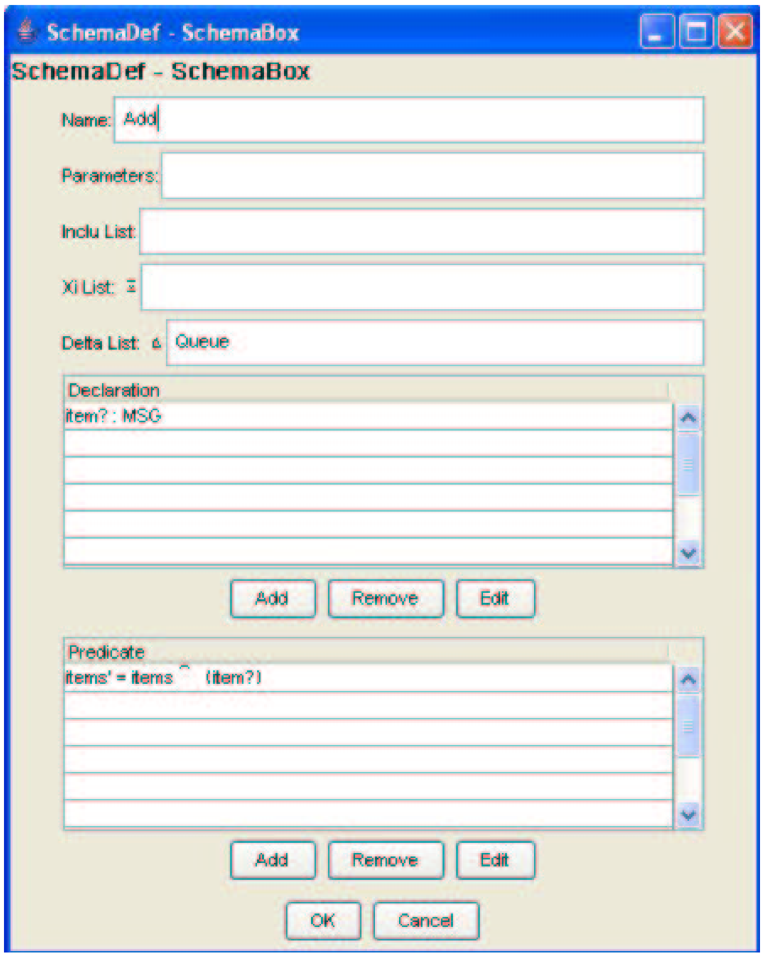


Figure 5.3: The component type

case tool. Then double click the *Add* operation schema in the graphical editor. A pop up frame will display the information of the component double-clicked. In this instance, it's a schema definition frame, as shown in figure 5.4. We can edit the name of this operation, the parameters, the including list, the Xi list, the delta list, the declarations and the predicates. When adding any declarations or predicates to the ZML file, an evaluator and validator will be used to check whether these new components conform to the XML schema and logically correct.

5.1.2 Query

This case tool supports five types of query for Z family specifications. Each type of query aims at providing some type of information on a specification, which we have described in detail in last chapter. See the figure 5.5.



SchemaDef - SchemaBox

Name: Add

Parameters:

Inclu List:

Xi List: Ξ

Delta List: Δ Queue

Declaration

item? : MSG

Predicate

items' = items ⁀ ⟨item?⟩

Figure 5.4: The edit of *Add* operation

Class Query

Class query aims at providing information on classes in the specification. One significant information user might want to know is the inheritance relationship among the classes in a TCOZ specification. In our case tool, we use a hierarchy chart to display this inheritance relationship. Another information user might want to know is the detail of a specified class. Take the *Queue* system specification for example again. After loading the *Queue* system specification into our case tool,

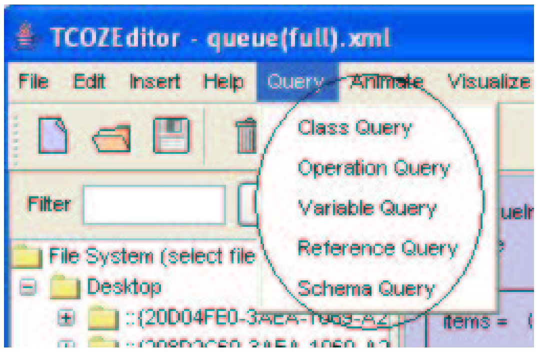


Figure 5.5: Five types of query

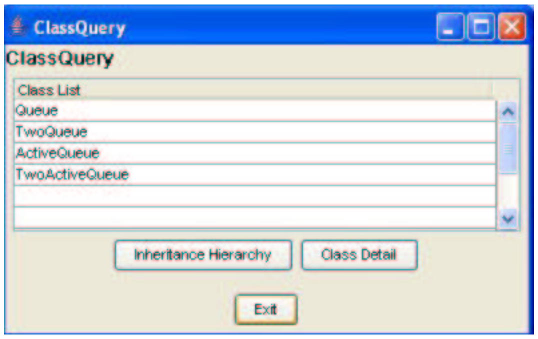


Figure 5.6: Class query popup frame

user can click the *Class Query* item in the menu shown in figure 5.5. Then a class query popup frame will display as in figure 5.6. At the beginning, this frame only display which classes this specification consists. We can see that there are four classes, *Queue*, *TwoQueue*, *ActiveQueue*, *TwoActiveQueue*, in this system. Click the *Inheritance Hierarchy* button, an inheritance hierarchy chart will pop up, as shown in figure 5.7. Select one class shown in figure 5.6, and click the *class detail* button, a class detail query frame will display, as shown in figure 5.8. In this frame, we list in detail which query user is allowed to ask about this class. Each item has

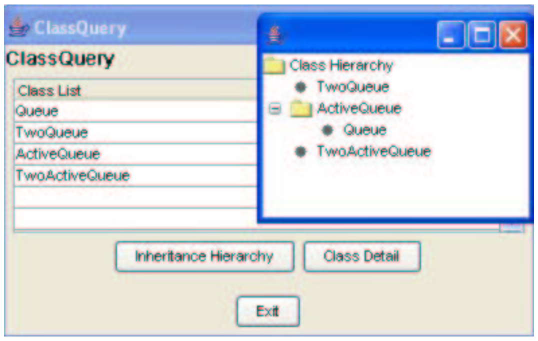


Figure 5.7: Inheritance hierarchy chart

a box on the left of it. Tick the box means that user wants the related information. Take the *ActiveQueue* class for example, we tick the boxes on the left of *Class Name*, *Superclass*, *State Var.* and *Operation* respectively. Then click “OK”. The result is shown as figure 5.9.

Schema Query

Schema query is the same as class query, aiming at providing information on schemas. A difference between class query and schema query is that there is no inheritance relationship among schemas. Click the *Schema Query* item in the menu, a schema query frame will pop up, as figure 5.10. A list of schemas in *Queue* specification is shown in this frame. We can also query which schema includes a specified schema or modifies it. In figure 5.10 we see that there are four schemas, *Queue*, *QueueInit*, *Add* and *Delete*, in this specification. And *Queue* is included by *QueueInit* and modified by *Add* and *Delete*.

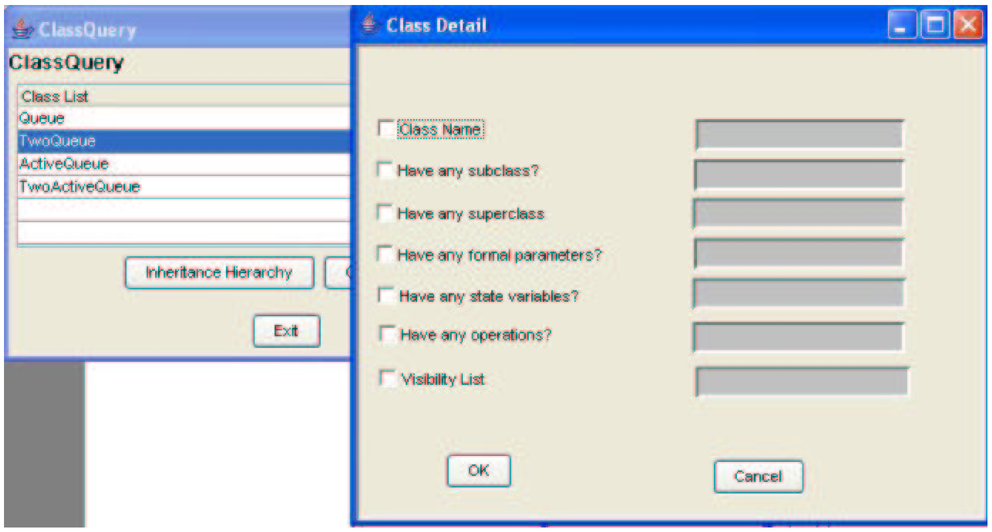
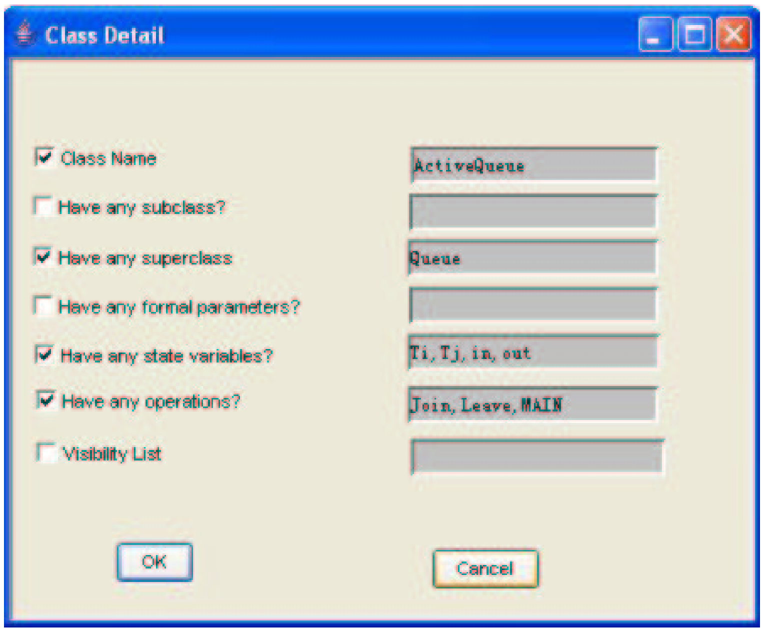


Figure 5.8: Class detail query popup frame

Operation Query

Operation query provides information on operations in a specification. After loading the ZML file successfully, take *Queue* system for instance again, click the *Operation Query* item in the menu, and an operation query frame will display, as shown in figure 5.11. All the operations in this specification are listed in this frame. The name of the classes they belong to and their visibility in these classes are also indicated. Selecting one specific operation in the list and clicking the *Operation Detail* button, the detail of this operation is display in another operation detail frame, see figure 5.12.

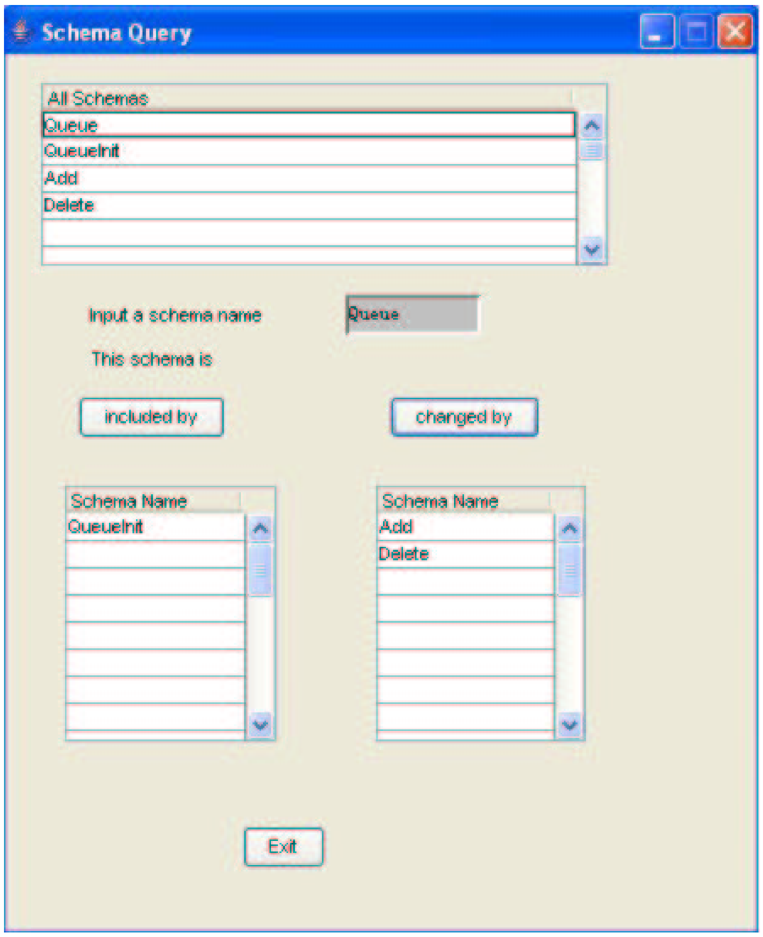


[✓] Class Name	ActiveQueue
[ ] Have any subclass?	
[✓] Have any superclass	Queue
[ ] Have any formal parameters?	
[✓] Have any state variables?	Ti, Tj, in, out
[✓] Have any operations?	Join, Leave, MAIN
[ ] Visibility List	

Figure 5.9: Class detail of *ActiveQueue*

Variable Query

The *Variable Query* frame provides information on all the variables in the specification. Click the *Variable Query* menu item and the *Variable Query* frame will pop up, as shown in figure 5.13. On the top of this frame are the name list of all the variables in *Queue* specification, the classes in which they are, and the operations in which they are (if any). On the bottom of this frame is the variable checking area. Input the name of a variable and the class in which it is, click the button *Query*, and the related information of this variable will be shown. This information includes: whether this variable is a state variable, whether it is visible (if it's in an operation), the type definition of it and its original form.



All Schemas
Queue
QueueInit
Add
Delete

Input a schema name: Queue

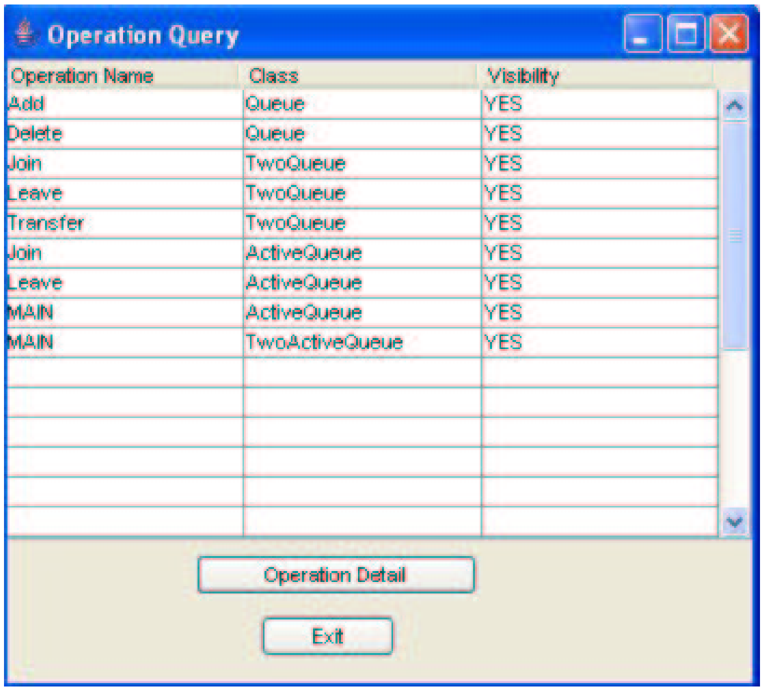
Schema Name
QueueInit

Schema Name
Add
Delete

Figure 5.10: Schema query frame

Reference Query

Reference query allows user to input a specific class name and query in which class this input class is referenced. See figure 5.14. We input *ActiveQueue* and click *Query* button. The result is, class *ActiveQueue* is referenced twice in class *TwoActiveQueue*. The variables that are the instance of *ActiveQueue* are *q1* and *q2*. *Reference Definition* gives the exact definitions of the reference.

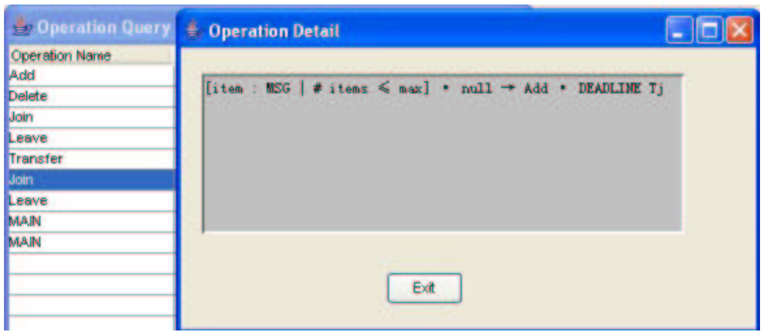


Operation Name	Class	Visibility
Add	Queue	YES
Delete	Queue	YES
Join	TwoQueue	YES
Leave	TwoQueue	YES
Transfer	TwoQueue	YES
Join	ActiveQueue	YES
Leave	ActiveQueue	YES
MAIN	ActiveQueue	YES
MAIN	TwoActiveQueue	YES

Figure 5.11: Operation query frame

5.1.3 Visualization

In this thesis we exploit Sun et al. [34, 35]'s UML projection to visualize the TCOZ specification. This approach transforms a TCOZ specification to diagrams. We link this transformation process to our case tool. Click the *Make Statechart* item in the menu, the process begins, as shown in figure 5.15 and figure 5.16. In figure 5.16, we can see that in the *Queue* specification, two classes *ActiveQueue* and *TwoActiveQueue* are transformed to XMI files. These XMI files can be shown in UML tool as statecharts. Figure 5.17 is the statechart of *ActiveQueue*.



Operation Name
Add
Delete
Join
Leave
Transfer
Join
Leave
MAIN
MAIN

$[item : MSG \mid \# items \leqslant max] \bullet null \rightarrow Add \bullet DEADLINE\ Tj$

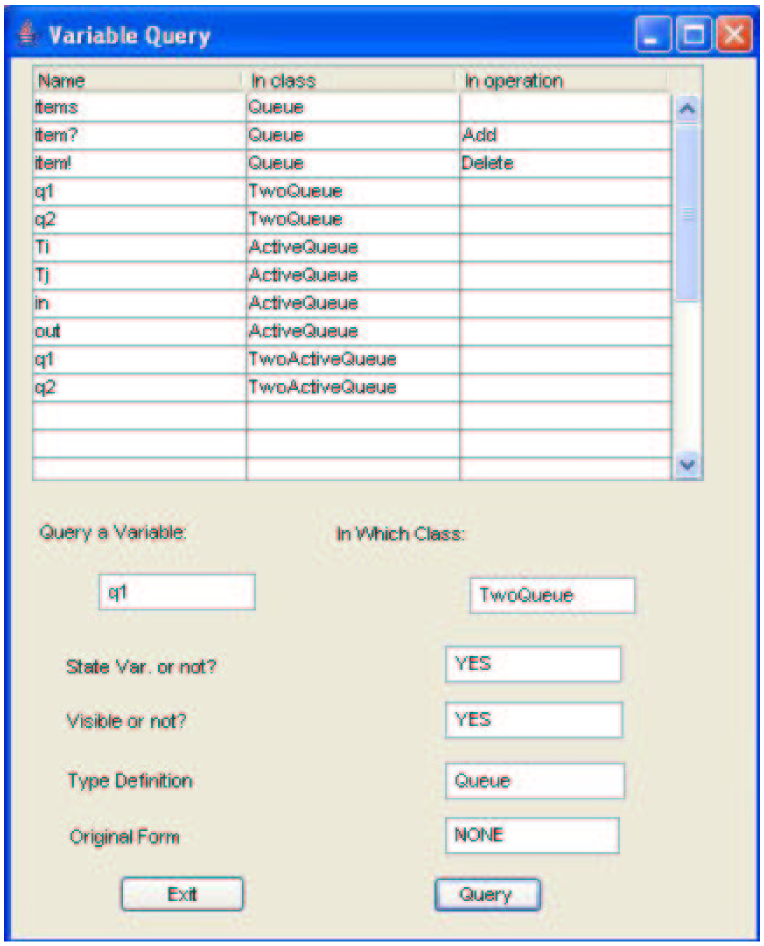
Figure 5.12: Operation detail frame

5.1.4 Animation

TCOZ Java Library

In last chapter we have introduced the translation rules of the animation from TCOZ to Java language. In this section, we will introduce the implementation and case study. As we discussed in last chapter, an equivalent library functions for handling specification constructs can greatly benefit the translation process from TCOZ to Java. Part of Java library to manipulate TCOZ constructs, i.e., set operations and channel declaration, is defined as follow.

```
%Set Definition%

public class SetDef{
   Vector content;

  public SetDef(Vector content){
          this.content= content;
   }


  public boolean isElement(String o){
   int size=this.content.size();
```



Name	In class	In operation
items	Queue	
item?	Queue	Add
item!	Queue	Delete
q1	TwoQueue	
q2	TwoQueue	
Ti	ActiveQueue	
Tj	ActiveQueue	
in	ActiveQueue	
out	ActiveQueue	
q1	TwoActiveQueue	
q2	TwoActiveQueue	

Query a Variable: q1

In Which Class: TwoQueue

State Var. or not? YES

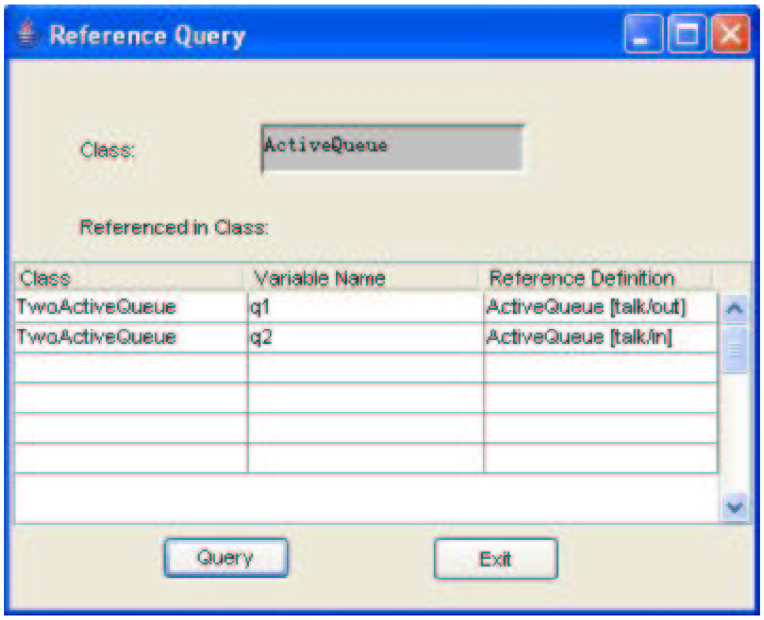
Visible or not? YES

Type Definition Queue

Original Form NONE

Figure 5.13: Variable query frame

```
    for(int i=0; i<size; i++){
    String temp= new String();
    temp= (String) this.content.elementAt(i);
    if(temp.equals(o)) return true;
    }
    return false;
     // o is a element of Vector this.content
    }

  public boolean isEmpty(){
     return this.content.isEmpty();
    }
 ......
}
```



Class: ActiveQueue

Referenced in Class:

Class	Variable Name	Reference Definition
TwoActiveQueue	q1	ActiveQueue [talk/out]
TwoActiveQueue	q2	ActiveQueue [talk/in]

Figure 5.14: Reference query frame

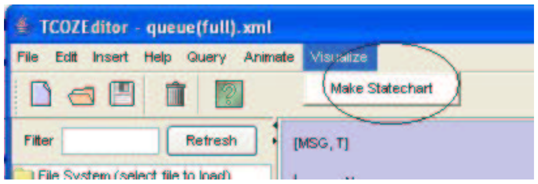
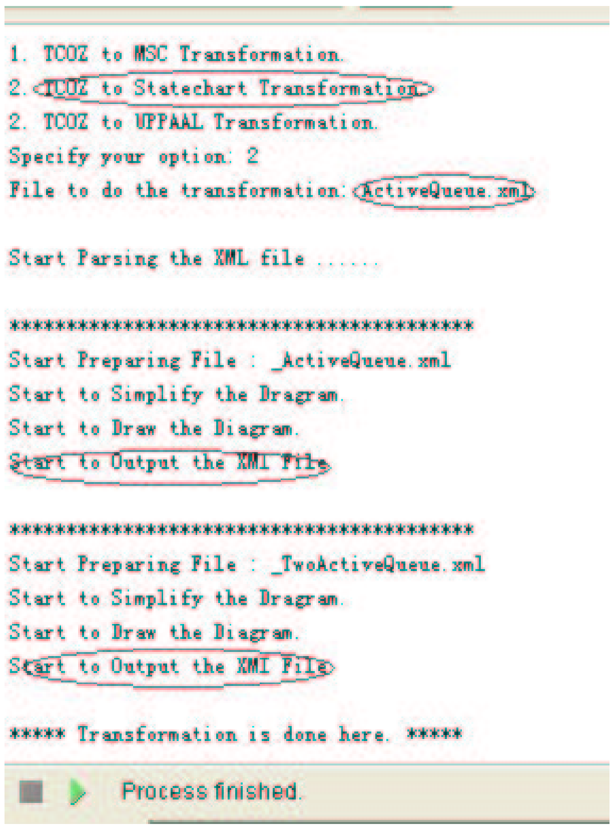


Figure 5.15: Click the *Make Statechart* button

```
%Channel Definition%

public class Signal extends Thread {
  public synchronized void wait(int T){
    for(int i=0;i<T;i++){}
      }
}

public class Write extends Thread {
   public void notifyIt(String S, int index,Signal sig){
        global.array[index]=S;
      synchronized(sig){
```



```
1. TCOZ to MSC Transformation.
2. TCOZ to Statechart Transformation.
2. TCOZ to UPPAAL Transformation.
Specify your option: 2
File to do the transformation: ActiveQueue.xml

Start Parsing the XML file ......

****************************************
Start Preparing File : _ActiveQueue.xml
Start to Simplify the Dragram.
Start to Draw the Diagram.
Start to Output the XMI File

****************************************
Start Preparing File : _TwoActiveQueue.xml
Start to Simplify the Dragram.
Start to Draw the Diagram.
Start to Output the XMI File

***** Transformation is done here. *****
```

Process finished.

Figure 5.16: The transform process from TCOZ to statecharts

```
         sig.notify();
       }
   }
}

public class Channel {
 int index;
 int Deadline;
 Signal sig;
 Write writeChannel;

  public Channel(int index, int T) {
    this.index= index;
    this.Deadline=T;
   }

  public void inChannel(String s){
      sig= new Signal();
```

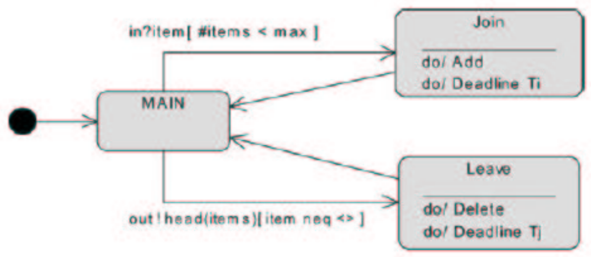
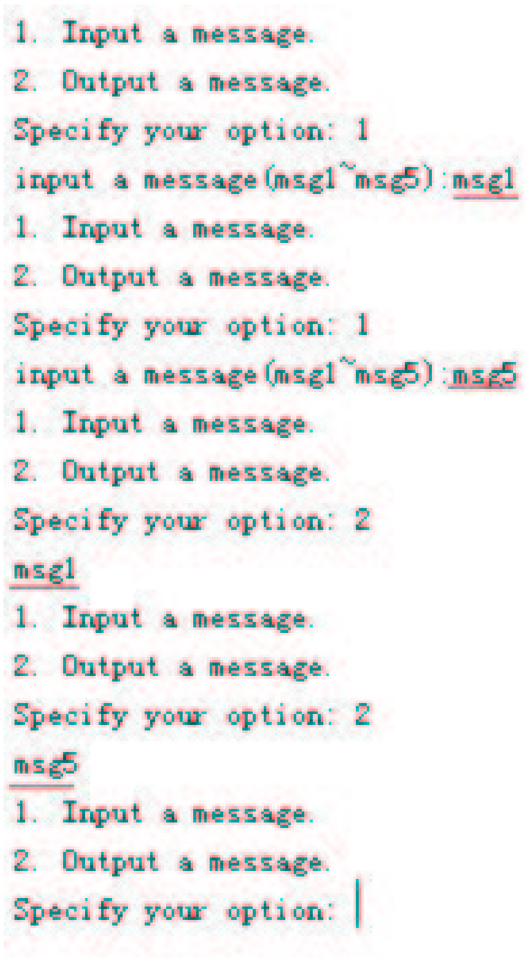


Figure 5.17: *ActiveQueue* statechart diagram

```
      writeChannel= new Write();
     sig.wait(Deadline);
     writeChannel.notifyIt(s, index, sig);
   }

  public boolean isChannelActive(){
     if(global.array[index].equals("")){
       return false;
      }else return true;
   }

  public String getChannel(){
     String s= new String();
     s=global.array[index];
     global.array[index]="";
     return s;
  }
}
```

Firstly a number of set functions such as subset, power set, union , intersection and so on are defined for matching the corresponding Z set constructs. To implement these we use a Java class *SetDef* to define a set construct, and then define the methods about a set, such as subset or power set, as the internal operations of this class. We have completed the entire Z set operations in Java. The first example above is part of them. The entire library can be found in Appendix A.



```
1. Input a message.
2. Output a message.
Specify your option: 1
input a message(msg1~msg5):msg1
1. Input a message.
2. Output a message.
Specify your option: 1
input a message(msg1~msg5):msg5
1. Input a message.
2. Output a message.
Specify your option: 2
msg1
1. Input a message.
2. Output a message.
Specify your option: 2
msg5
1. Input a message.
2. Output a message.
Specify your option:
```

Figure 5.18: Animation

Secondly, TCOZ communication constructs such as channel, sensor and actuator are all implemented as Java classes. The second example above shows a TCOZ channel. In each channel, two thread classes *Signal* and *Write* provide a guarantee for the time limitation mechanism of channel and achieve the synchronization.

TCOZ Java Projection

To project TCOZ specification to Java, XSL Transformation is applied. So far we have implemented a transformation of *Queue* system. This projection is suitable for any form of system "Queue". The change of the name of any classes, operations or variables is allowed and won't affect the producing of the program (the variables of program will change accordingly).

The segment of the XSL stylesheet is as follow.

```
<?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet
version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text"/>

<xsl:template match="/">
    <xsl:apply-templates select="//classDef"/>
</xsl:template>

<xsl:template match="classDef"> ... </xsl:template>

<xsl:template match="operation"> ... </xsl:template>

</xsl:stylesheet>
```

The above XSL transformation states that a projection will be made on each defined TCOZ class in XML to construct their corresponding Java classes. For example,the operations are captured through the *operation* tag. The entire XSLT file can be found in Appendix B.

The case study

Consider the class *ActiveQueue* and *TwoActiveQueue*in the *Queue* system, the frame of their translated specification in Java is as follow. The entire Java classes can be found in Appendix C.

```
%ActiveQueue Class%

class ActiveQueue extends Queue {
         ......
    public ActiveQueue(int inNo, int outNo){
      ......
      }
    public void Join(String item){
        ......
        }
    public String Leave(){
       ......
       }
    public void Transfer(){
          ......
           }
      }
}

%TwoActiveQueue Class%

class TwoActiveQueue {
      ......
    public TwoActiveQueue(){
       }
    public void transfer(){
       ......
       }
    public static void main(String args[]) {
      ......
     }
}
```

After translating *ActiveQueue* and *TwoActiveQueue* into Java classes, we can put them with the *Channel* class and *SetDef* class together and run it in Java running environment. Figure 5.18 is the result. We attempt a simple test of the First In First Out(FIFO) property of *TwoActiveQueue*. The input message is limited to the type of *MSG*(msg1 $\sim$ msg5). We firstly input message *msg1* (type of *MSG*) into *TwoActiveQueue*. Message *msg5* is the second input message. Then we output messages from *TwoActiveQueue*. We can find that *msg1* is firstly output, *msg5* secondly. The sequence of output is the same as the one of input.

Chapter 6

Conclusions and Future Work

This chapter concludes the whole thesis and proposes several directions for future research works.

6.1 Conclusions

Along with the rapid development of formal methods techniques and specification languages, the requirement of understanding a specification rises. We believe that specification comprehension may be viewed as a new research area. Different from some recent works of specification comprehension which mainly focus on the comprehensibility of a specification, in this thesis, we attempt to carry out works similar to program understanding and attempt to provide some techniques to help users understand the specification correctly and quickly.

In this thesis, we propose a framework of specification comprehension for Z family, attempting to utilize some techniques to illustrate the static and dynamic properties of a Z family specification. The techniques we introduce in this thesis include query, visualization and animation. The environment of Z family we exploit here is ZML.

The query of Z family specifications is similar to the query of a program, including class query, schema query, operation query, variable query and cross-reference query. Query tool focuses on displaying the static properties of a specification.

The visualization of Z family specifications aims at improving the readability and interpretability of a specification. In this thesis we introduce an approach of visualization utilizing UML projection proposed by Sun et al.

The animation of a Z family specification helps performing the dynamic properties of a specification. Although some dynamic properties can be proved using proving techniques, they are more apparent after the specification is animated. In this thesis, we introduce the animation from TCOZ to Java.

A case tool is presented at the end of this thesis. This case tool provides an environment for displaying and editing Z family specifications. In this case tool user can query information on classes, schemas, etc., of a specification. We also link the UML projection process to this case tool. The animation process from TCOZ to Java along with the case study is implemented in this case tool as well. This case tool also supports simple logic and semantic checking.

6.2 Future Works

We propose some future work directions related to specification comprehension as follows:

- The query of TCOZ can be extended by answering more complex queries, especially the cross-reference query part. In our implementation, the reference query only relates to the cross-references between classes. In future work, reference between specifications, even between systems, should be included. Other more complex queries such as query about containment, query about inheritance, query about polymorphic operation, etc., should be added.
- In future work, we can attempt a query of TCOZ while visualizing it. That is, combine the query part and visualization part together.
- In future work, an entire animation tool should be developed. This tool should be able to transform any TCOZ specifications into Java programs

automatically and thoroughly. The idea of XVCL (XML-based Variant Configuration Language) [40] may be helpful to improve this tool. XVCL is a meta-programming technique and tool that provides effective reuse mechanisms. Our future animation tool may exploit these reuse mechanisms to achieve high efficiency.

- The environment of TCOZ specification in this thesis is ZML. In future work, we will try to base our specification specification on Semantic Web, proposed by Dong et al [7].

- In future work, besides query, visualization and animation, more tools should be included to extend the TCOZ comprehension.

Bibliography

[1] J-R. Abrial. *The B-Book*. Cambridge University Press, 1996.

[2] V. R. Basili and H. D. Mills. Understanding and documenting programs. *IEEE Transactions on Software Engineering*, SE-8(3):270 – 283, March 1982.

[3] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14:25–59, 1987.

[4] R. Brooks. Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies*, 18(6):543 – 554, July 1983.

[5] World Wide Web Consortium(W3C). Document object model(dom). http://www.w3c.org/DOM/.

[6] J. S. Dong, Y. F. Li, J. Sun, J. Sun, and H. Wang. Xml-based static type checking and dynamic visualization for tcoz. *ICFEM'02*, pages 311–322, Oct 2002.

[7] J. S. Dong, J. Sun, and H. Wang. Semantic Web for Extending and Linking Formalisms. Technical Report TRB4/02, School of Computing, National University of Singapore, March 2002.

[8] R. Duke and G. Rose. *Formal Object Oriented Specification Using Object-Z*. Cornerstones of Computing, (series editors: R. Bird and C.A.R Hoare). Macmillan, March 2000.

[9] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1*, volume 6 of *EATCS Monographs on Theoretical Comput. Sci.* Springer-Verlag, 1985.

[10] K. Finney, N. Fenton, and A. Fedorec. Effects of structure on the comprehensibility of formal specifications. In *IEE Proceeding of Software,146(4)*, pages 193–202, 1999.

[11] J. Goguen. OBJ as a theorem prover. Technical report, 1988.

[12] I. Hayes and C. Jones. Specifications are not (necessarily) executable. *Software Eng. Journal*, 4(6):330–339, November 1989.

[13] I. J. Hayes and B. P. Mahony. Using units of measurement in formal specifications. *Formal Aspects of Computing*, 7(3), 1995.

[14] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.

[15] J. Horning. Combining algebraic and predicative specifications in Larch. In H. Ehrig, C. Floyd, M. Nivat, and J. Thatcher, editors, *TAPSOFT'85 (part*

II): Formal Methods and Software Development, volume 186 of *Lect. Notes in Comput. Sci.*, pages 12–26. Springer-Verlag, 1985.

[16] C. Hung. CCS used as a proof-assistant tool. In M. Diaz, editor, *Protocol Specification, Testing, and Verification, V*, pages 387–398. North-Holland, 1986.

[17] International Organization for Standardization, Geneva. *Units of measurement: handbook on international standards for units of measurement*, 1979.

[18] S. Jarzabek. Design of Flexible Static Program Analyzers with PQL. *IEEE Trans. Software Eng.*, pages 197–215, March 1998.

[19] Judith Kabeli and Peretz Shoval. Foom and opm methodologies - experimental comparison of user comprehension. In *INGITS 2002*, pages 107–122.

[20] S. Letovsky. Cognitive processes in program comprehension, empirical studies of programmers. In *Albex, Norwood NJ*, pages 58 – 79, 1986.

[21] S. Letovsky and E. Soloway. Delocalized plans and program comprehension. *IEEE Software*, 19(3):41–48, May 1986.

[22] D. C. Littman, J. Pinto, S. Letovsky, and E. Soloway. Mental models and software maintenance, empirical studies of programmers. In *Albex, Norwood NJ*, pages 80 – 98, 1986.

[23] B. Mahony and J. S. Dong. Blending Object-Z and Timed CSP: An introduction to TCOZ. Technical Report 97-22, Mathematical and Informa-

tion Sciences, Commonwealth Scientific and Industrial Research Organisation (CSIRO), Australia, 1997.

[24] B. Mahony and J. S. Dong. Timed Communicating Object Z. Technical Report 98-37, Mathematical and Information Sciences, Commonwealth Scientific and Industrial Research Organisation (CSIRO), Australia, 1998.

[25] B. Mahony and J. S. Dong. Overview of the semantics of TCOZ. In K. Araki, A. Galloway, and K. Taguchi, editors, *IFM'99: Integrated Formal Methods, York, UK*, pages 66–85. Springer-Verlag, June 1999.

[26] B. Mahony and J. S. Dong. Sensors and Actuators in TCOZ. In J. Wing, J. Woodcock, and J. Davies, editors, *FM'99: World Congress on Formal Methods*, Lect. Notes in Comput. Sci., pages 1166–1185, Toulouse, France, September 1999. Springer-Verlag.

[27] B. P. Mahony and J. S. Dong. Network topology and a case-study in TCOZ. In *ZUM'98 The* 11^{th} *International Conference of Z Users*. Springer-Verlag, September 1998.

[28] M. Peleg and D. Dori. The model multiplicity problem : Experimenting with realtime specification methods. In *IEEE Trans. on Soft. Eng.,Vol. 26, No. 8*, pages 742–759.

[29] S. Schneider, J. Davies, D. M. Jackson, G. M. Reed, J. N. Reed, and A. W. Roscoe. Timed CSP: Theory and practice. In J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*,

volume 600 of *Lect. Notes in Comput. Sci.*, pages 640–675. Springer-Verlag, 1992.

[30] B. Shneiderman and R. Mayer. Syntactic / semantic interactions in programmer behaviour: A model and experimental results. *International Journal of Computer and Information Sciences*, 8(3):219 – 238, 1979.

[31] C. Snook and R.Harrison. Experimental comparison of the comprehensibility of a z specification and its implementation. In *Proceedings of EASE 2001: Papers from The Conference on Empirical Assessment In Software Engineering*, 2001.

[32] E. Soloway, B. Adelson, and K. Ehrlich. Knowledge and processes in the comprehension of computer programs. In R. Glaser M. Chi and M. Farr, editors, *The Nature of Expertise*, pages 129 – 152. A. Lawrence Erlbaum Associates, Hillside, NJ, 1988.

[33] E. Soloway and K. Ehrlich. Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*, SE-10(5):595 – 609, September 1984.

[34] J. Sun, J. S. Dong, J. Liu, and H. Wang. A XML/XSL Approach to Visualize and Animate TCOZ. In *The 8th Asia-Pacific Software Engineering Conference (APSEC'01)*, pages 453–460. IEEE Press, 2001.

[35] J. Sun, J. S. Dong, J. Liu, and H. Wang. Object-Z Web Environment and Projections to UML. In *WWW-10: 10th International World Wide Web Conference*, pages 725–734. ACM Press, May 2001.

[36] J. Sun, J. S. Dong, J. Liu, and H. Wang. A formal object approach to the design of zml. *Annals of Software Engineering, an international journal*, 2002. (accepted).

[37] S. R. Tilley, S. Paul, and D. B. Smith. Towards a framework for program understanding. In *Proc. of the 4th International Workshop on Program Comprehension (IWPC'96)*, 1996.

[38] A. von Mayrhauser and A. M. Vans. Program understanding - a survey. Technical report, August 1994.

[39] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof.* Prentice-Hall International, 1996.

[40] XVCL Team, School of Computing, National University of Singapore. XVCL (XML-based Variant Configuration Language). http://fxvcl.source.forge.net/.

Appendix A

Library Functions For Animation

A.1 Set Library Class

```
%Set Definition%

public class SetDef{
   Vector content;


  public SetDef(Vector content){
          this.content= content;
}


public boolean isElement(String o){
   int size=this.content.size();
   for(int i=0; i<size; i++){
   String temp= new String();
   temp= (String) this.content.elementAt(i);
   if(temp.equals(o)) return true;
   }
   return false;
    // o is a element of Vector this.content

}
```

```
  public boolean isEmpty(){
     return this.content.isEmpty();
}

  public Vector getContent(){
     return this.content;
}

  public boolean append(Object o){
     int s=this.content.size();
     for(int i=0;i<s;i++){
      if(content.elementAt(i).equals(o)) return false;
  }
      this.content.add(o);
      return true;
}

  public boolean isSubsetof(SetDef A){
     return A.getContent().containsAll(this.content);
}

  public boolean equals(SetDef A){
     return this.content.equals(A.getContent());
}


  public SetDef powerSet(){
   SetDef powerSet= new SetDef(null);
   int size= this.content.size();
   Vector temp=new Vector();
   temp=null;
   powerSet.getContent().add(temp);
    for(int i=0;i<size;i++)
     for(int j=i;j<size;j++)
       for(int k=i;k<=j;k++)
        { temp.add(this.content.elementAt(k));
          powerSet.getContent().add(temp);
        }
   return powerSet;
}

  public SetDef CartProdwith(SetDef A){
    Vector a=new Vector();
    a=A.getContent();
    int s1=this.content.size();
    int s2=a.size();
    Vector cartprod= new Vector();
    Vector temp=new Vector(2);
    for(int i=0;i<s1;i++)
      for(int j=0;j<s2;j++)
         {  temp.add(this.content.elementAt(i));
            temp.add(a.elementAt(j));
            cartprod.add(temp);
         }
    SetDef cartprodset= new SetDef(cartprod);
    return cartprodset;
}


  public SetDef Unionwith(SetDef A){
       Vector a= new Vector();
       a=A.getContent();
       Vector union= new Vector();
       Vector intersect =new Vector();
       union.add(this.content);
       intersect.add(this.content);
       intersect.retainAll(a);
       a.removeAll(intersect);
       union.add(a);
       SetDef unionset= new SetDef(union);
       return unionset;
}


   public SetDef Intersectwith(SetDef A){
       Vector a= new Vector();
       a=A.getContent();
       Vector intersect= new Vector();
       intersect.add(this.content);
       intersect.retainAll(a);
       SetDef intersectset= new SetDef(intersect);
       return intersectset;
}


   public SetDef Substractwith(SetDef A){
       Vector a= new Vector();
       a=A.getContent();
       Vector sub= new Vector();
       Vector intersect= new Vector();
       intersect.add(this.content);
       intersect.retainAll(a);
       sub.add(this.content);
       sub.removeAll(intersect);
       SetDef subset= new SetDef(sub);
       return subset;
}


   public int Cardinality(){
       return this.content.size();
}


}
```

A.2 Channel and Sensor/Actuator

```
%Channel Definition%

public class global {

static String[] array= new String[6];

}


public class Signal extends Thread {
  public synchronized void wait(int T){
    for(int i=0;i<T;i++){}
      }
}


public class Write extends Thread {
   public void notifyIt(String S, int index,Signal sig){
        global.array[index]=S;
      synchronized(sig){
         sig.notify();
       }
   }
}


public class Channel {
 int index;
 int Deadline;
 Signal sig;
 Write writeChannel;

  public Channel(int index, int T) {
    this.index= index;
    this.Deadline=T;
}



  public void inChannel(String s){
      sig= new Signal();
      writeChannel= new Write();
     sig.wait(Deadline);
     writeChannel.notifyIt(s, index, sig);
}

  public boolean isChannelActive(){
  if(global.array[index].equals("")){
  return false;
  }else return true;
  }

  public String getChannel(){
  String s= new String();
  s=global.array[index];
  global.array[index]="";
  return s;
  }

}

%Sensor and Actuator%

public class SensorActuator{
   int index;

public SensorActuator(int index){
   this.index=index;
   global.array[index]="";
}

public void update(String s){
   global.array[this.index]=s;
}

public String read(){
   return global.array[this.index];
}

}
```

Appendix B

The Entire XSLT File for *Queue* System

```
<?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet
version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text"/>

<xsl:template match="/">
    <xsl:apply-templates select="//classDef"/>
</xsl:template>

<xsl:template match="classDef">
    <xsl:if test="@a[.='c1']">
    <xsl:text> public class </xsl:text>
    <xsl:value-of select="name"/>
    <xsl:text> {
      </xsl:text>
      <xsl:text>
        Vector msgs;
      </xsl:text>
    <xsl:text> SetDef </xsl:text>
    <xsl:value-of select="//basicTypeDef/name"/>
    <xsl:text>;
      </xsl:text>
      <xsl:text> Vector </xsl:text> <xsl:value-of select=
                      "state/declaration/variable"/>
      <xsl:text>;
      int head=0;
      int tail=0;
      </xsl:text>

      <xsl:text> public </xsl:text>
      <xsl:value-of select="name"/>
      <xsl:text>()
      { msgs = new Vector();
        msgs.add("msg1");
        msgs.add("msg2");
        msgs.add("msg3");
        msgs.add("msg4");
                msgs.add("msg5");
        </xsl:text>
        <xsl:value-of select="//basicTypeDef/name"/>
         <xsl:text> = new SetDef(msgs);
        </xsl:text>
        <xsl:value-of select="state/declaration/variable"/>
        <xsl:text>= new Vector(</xsl:text>
        <xsl:value-of select="//axiomaticDef/predicate/expression/number"/>
        <xsl:text>);
        }
        </xsl:text>

      <xsl:apply-templates select="operation"/>


      <xsl:text>
    }
      </xsl:text>
    </xsl:if>




    <xsl:if test="@a[.='c3']">
    <xsl:text>class </xsl:text>
    <xsl:value-of select="name"/>
    <xsl:text> extends </xsl:text>
    <xsl:value-of select="inheritedClass/name"/>
    <xsl:text> {
      </xsl:text>

      <xsl:for-each select="state/declaration">
```

```
          <xsl:if test="dataType/type[.='T']">
          <xsl:for-each select="variable">
           <xsl:text> int </xsl:text> <xsl:value-of select="."/>
           <xsl:text>=100;
           </xsl:text>
          </xsl:for-each>
          </xsl:if>
          <xsl:if test="dataType/type[.='Chan']">
          <xsl:for-each select="variable">
           <xsl:text> Channel </xsl:text> <xsl:value-of select="."/>
           <xsl:text>;
          </xsl:text>
          </xsl:for-each>
          </xsl:if>
      </xsl:for-each>
      int inNo,outNo;
<xsl:text>

      </xsl:text>
      <xsl:text> public </xsl:text> <xsl:value-of select="name"/>
      <xsl:text>(int inNo, int outNo){this.inNo=inNo; this.outNo=outNo;}
      </xsl:text>
    <xsl:apply-templates select="operation"/>
    <xsl:text>
}
      </xsl:text>
    </xsl:if>

    <xsl:if test="@a[.='c4']">
    <xsl:text>class </xsl:text>
    <xsl:value-of select="name"/>
    <xsl:text> { </xsl:text>
    <xsl:text>
    </xsl:text>
      <xsl:text> int talkNo=1;
      int inNo=0;
      int outNo=2;

      </xsl:text>

       <xsl:value-of select="state/declaration1/dataType/type"/>
       <xsl:text> </xsl:text>
       <xsl:value-of select="state/declaration1/variable"/>
       <xsl:text>= new </xsl:text>
       <xsl:value-of select="state/declaration1/dataType/type"/>
```

```
        <xsl:text>(inNo, talkNo);
            </xsl:text>
        <xsl:value-of select="state/declaration2/dataType/type"/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="state/declaration2/variable"/>
        <xsl:text>= new </xsl:text>
         <xsl:value-of select="state/declaration2/dataType/type"/>
         <xsl:text>(talkNo, outNo);
            </xsl:text>

        <xsl:text>
        public </xsl:text><xsl:value-of select="name"/><xsl:text>(){
        }

        </xsl:text>

     <xsl:apply-templates select="operation"/>
     <xsl:text>

     public static void main(String args[]) {
       </xsl:text>
      <xsl:value-of select="name"/> <xsl:text> myTAQ = new </xsl:text>
      <xsl:value-of select="name"/><xsl:text>();
        myTAQ.transfer();
        </xsl:text>

<xsl:text>
     }


}


      </xsl:text>
    </xsl:if>

</xsl:template>


<xsl:template match="operation">

 <xsl:if test="@a[.='o1']">
        <xsl:text> public void </xsl:text>
        <xsl:value-of select="name"/>
        <xsl:text>(String </xsl:text>
         <xsl:value-of select="declaration/variable"/>
        <xsl:text>){
        </xsl:text>
        <xsl:text>
        if(this.</xsl:text>
        <xsl:value-of select="//basicTypeDef/name"/>
        <xsl:text>.isElement(</xsl:text>
        <xsl:value-of select="declaration/variable"/>
        <xsl:text>))
        </xsl:text>
        <xsl:text>this.</xsl:text><xsl:value-of select="deltaList"/>
        <xsl:text>.add(</xsl:text>
         <xsl:value-of select="declaration/variable"/> <xsl:text>);
        this.tail++;
        </xsl:text>
        <xsl:text>
        }


       </xsl:text>
      </xsl:if>

       <xsl:if test="@a[.='o2']">
        <xsl:text> public String </xsl:text>
        <xsl:value-of select="name"/> <xsl:text>(){
        </xsl:text>
        <xsl:text> String </xsl:text>
        <xsl:value-of select="declaration/variable"/> <xsl:text>;
        </xsl:text>
        <xsl:value-of select="declaration/variable"/>
        <xsl:text> = (String) </xsl:text>
        <xsl:text>this.</xsl:text><xsl:value-of select="deltaList"/>
        <xsl:text>.elementAt(head);
        this.head++;
        return </xsl:text> <xsl:value-of select="declaration/variable"/>
         <xsl:text>;
        }

        </xsl:text>
      </xsl:if>

       <xsl:if test="@a[.='o6']">
        <xsl:text> public void </xsl:text>
        <xsl:value-of select="name"/>
        <xsl:text>(String </xsl:text>
        <xsl:value-of select="processExpr/guard/declaration/variable"/>
         <xsl:text>){
        </xsl:text>
        <xsl:text> String temp;
        </xsl:text>
        <xsl:value-of select="processExpr/processExpr/event"/>
        <xsl:text>= new Channel(inNo,</xsl:text>
        <xsl:value-of select="processExpr/processExpr/processExpr
                                                  /deadline"/>
        <xsl:text>);
        </xsl:text>
        <xsl:value-of select="processExpr/processExpr/event"/>
        <xsl:text>.inChannel(</xsl:text>
        <xsl:value-of select="processExpr/guard/declaration/variable"/>
        <xsl:text>);
        </xsl:text>
      <xsl:text>if(</xsl:text>
      <xsl:value-of select="processExpr/processExpr/event"/>
      <xsl:text>.isChannelActive())
        </xsl:text>
        <xsl:text>temp = </xsl:text>
        <xsl:value-of select="processExpr/processExpr/event"/>
        <xsl:text>.getChannel();
        </xsl:text>
        <xsl:value-of select="processExpr/processExpr/processExpr
                                   /processExpr/simpleProExp"/>
        <xsl:text>(</xsl:text>
        <xsl:value-of select="processExpr/guard/declaration/variable"/>
         <xsl:text>);
         }

        </xsl:text>

        </xsl:if>


<xsl:if test="@a[.='o7']">
        <xsl:text> public String </xsl:text>
         <xsl:value-of select="name"/> <xsl:text>(){
        </xsl:text>
        <xsl:text> String temp;
        </xsl:text>
        <xsl:value-of select="processExpr/processExpr/event"/>
        <xsl:text>= new Channel(outNo,</xsl:text>
        <xsl:value-of select="processExpr/processExpr/processExpr
```

```
                                                          /deadline"/>
       <xsl:text>);
       </xsl:text>
       <xsl:text> temp=</xsl:text>
       <xsl:value-of select="processExpr/processExpr/processExpr/
                    processExpr/simpleProExp"/>
       <xsl:text>();
       </xsl:text>
       <xsl:value-of select="processExpr/processExpr/event"/>
       <xsl:text>.inChannel(temp);
       </xsl:text>
     <xsl:text>if(</xsl:text>
     <xsl:value-of select="processExpr/processExpr/event"/>
     <xsl:text>.isChannelActive())
       </xsl:text>
       <xsl:text>temp = </xsl:text>
       <xsl:value-of select="processExpr/processExpr/event"/>
        <xsl:text>.getChannel();
       </xsl:text>
       <xsl:text> return temp;
       }

       </xsl:text>

       </xsl:if>


<xsl:if test="name[.='MAIN']">
       <xsl:choose>
         <xsl:when test="processExpr/processExpr/
                 processExpr/proConnSym[.='externalChoice']">
           <xsl:text> public void Transfer(){
           String messageIn= new String();
           String messageOut= new String();
           while(true){
                BufferedReader stdin= new
                   BufferedReader(new InputStreamReader(System.in));
                System.out.print("input a message(msg1~msg5):");
               try{
                 messageIn= stdin.readLine();
               }catch (Exception e)
                 {System.err.println("cannot get a message!");}
           </xsl:text>
           <xsl:value-of select="processExpr/processExpr/
           processExpr/processExpr1/simpleProExp"/>
```

```
          <xsl:text>(messageIn);
          messageOut= </xsl:text><xsl:value-of select="processExpr/
          processExpr/processExpr/processExpr2/simpleProExp"/>
          <xsl:text>();
          System.out.println(messageOut);
          }
}
          </xsl:text>
        </xsl:when>
        <xsl:otherwise>
          <xsl:text> public void transfer(){
          String messageIn=new String();
          String messageTalk=new String();
          String messageOut=new String();
          String option= new String();
          BufferedReader stdin=
               new BufferedReader(new InputStreamReader(System.in));

         while(true)
         {

         System.out.println ("1. Input a message.");
       System.out.println ("2. Output a message.");
         System.out.print ("Specify your option: ");
         try{
         option = stdin.readLine();
         }catch (Exception e)
          {System.err.pringln("cannot get an option!");}
          if (option.equals("1"))
     {
          System.out.print("input a message(msg1~msg5):");
            try{
             messageIn= stdin.readLine();
            }catch (Exception e)
          {System.err.pringln("cannot get a message!");}

         </xsl:text>

         <xsl:value-of select="processExpr/activeObject1"/>
         <xsl:text>.</xsl:text>
         <xsl:for-each select="//classDef">
          <xsl:if test="@a[.='c3']">
            <xsl:for-each select="operation">
             <xsl:if test="name[.='MAIN']">
               <xsl:value-of select="processExpr/processExpr/
```

```
      processExpr/processExpr1/simpleProExp"/>
      <xsl:text>(messageIn);
      </xsl:text>
    </xsl:if>
   </xsl:for-each>
 </xsl:if>
</xsl:for-each>

<xsl:text>messageTalk =</xsl:text><xsl:value-of
select="processExpr/activeObject1"/><xsl:text>.</xsl:text>
<xsl:for-each select="//classDef">
 <xsl:if test="@a[.='c3']">
   <xsl:for-each select="operation">
    <xsl:if test="name[.='MAIN']">
      <xsl:value-of select="processExpr/processExpr/
      processExpr/processExpr2/simpleProExp"/><xsl:text>();
      </xsl:text>
    </xsl:if>
   </xsl:for-each>
 </xsl:if>
</xsl:for-each>

<xsl:value-of select="processExpr/activeObject2"/>
<xsl:text>.</xsl:text>
<xsl:for-each select="//classDef">
 <xsl:if test="@a[.='c3']">
   <xsl:for-each select="operation">
    <xsl:if test="name[.='MAIN']">
      <xsl:value-of select="processExpr/processExpr/
      processExpr/processExpr1/simpleProExp"/>
      <xsl:text>(messageTalk);
      </xsl:text>
    </xsl:if>
   </xsl:for-each>
 </xsl:if>
</xsl:for-each>

<xsl:text>}

   if (option.equals("2"))
{
<xsl:text>

<xsl:text>messageOut =</xsl:text><xsl:value-of
select="processExpr/activeObject2"/><xsl:text>.</xsl:text>
```

```
          <xsl:for-each select="//classDef">
           <xsl:if test="@a[.='c3']">
             <xsl:for-each select="operation">
              <xsl:if test="name[.='MAIN']">
                <xsl:value-of select="processExpr/processExpr/
                processExpr/processExpr2/simpleProExp"/><xsl:text>();
                </xsl:text>
              </xsl:if>
             </xsl:for-each>
           </xsl:if>
          </xsl:for-each>

          <xsl:text> System.out.println(messageOut);
         }
} }
          </xsl:text>

         </xsl:otherwise>
       </xsl:choose>
     </xsl:if>




</xsl:template>

</xsl:stylesheet>
```

Appendix C

Java Classes of *Queue*, *ActiveQueue* and *TwoActiveQueueClasses*

```
%Queue Class%

public class Queue {

        Vector msgs;
        SetDef MSG;
        Vector items;
        int head=0;
        int tail=0;

    public Queue()
      { msgs = new Vector();
        msgs.add("msg1");
        msgs.add("msg2");
        msgs.add("msg3");
        msgs.add("msg4");
                msgs.add("msg5");
        MSG = new SetDef(msgs);
        items= new Vector(100);
        }

    public void Add(String itemIn){
       if(this.MSG.isElement(itemIn))
       this.items.add(itemIn);
       this.tail++;

       }

    public String Delete(){
        String itemOut;
       itemOut = (String) this.items.elementAt(head);
       this.head++;
       return itemOut;
       }

 }



%ActiveQueue Class%

class ActiveQueue extends Queue {

       int Ti=100;
       int Tj=100;
       Channel in;
       Channel out;
       int inNo,outNo;

   public ActiveQueue(int inNo, int outNo){
      this.inNo=inNo;
      this.outNo=outNo;
      }

   public void Join(String item){
        String temp;
       in= new Channel(inNo,Tj);
       in.inChannel(item);
       if(in.isChannelActive())
       temp = in.getChannel();
       Add(item);
        }

   public String Leave(){
       String temp;
       out= new Channel(outNo,Ti);
       temp=Delete();
       out.inChannel(temp);
       if(out.isChannelActive())
       temp = out.getChannel();
       return temp;
       }

   public void Transfer(){
           String messageIn= new String();
           String messageOut= new String();
           while(true){
                BufferedReader stdin=
                   new BufferedReader(new InputStreamReader(System.in));
                System.out.print("input a message(msg1~msg5):");
               try{
                 messageIn= stdin.readLine();
               }catch (Exception e)
                 {System.err.println("cannot get a message!");}
           Join(messageIn);
           messageOut= Leave();
           System.out.println(messageOut);
           }
      }

}


%TwoActiveQueue Class%

class TwoActiveQueue {
      int talkNo=1;
      int inNo=0;
      int outNo=2;

      ActiveQueue q1= new ActiveQueue(inNo, talkNo);
      ActiveQueue q2= new ActiveQueue(talkNo, outNo);

       public TwoActiveQueue(){
       }

   public void transfer(){
           String messageIn=new String();
           String messageTalk=new String();
           String messageOut=new String();
           String option= new String();
           BufferedReader stdin= new
```

```
            BufferedReader(new InputStreamReader(System.in));

        while(true)
        {

        System.out.println ("1. Input a message.");
      System.out.println ("2. Output a message.");
        System.out.print ("Specify your option: ");
        try{
        option = stdin.readLine();
        }catch (Exception e)
         {System.err.pringln("cannot get an option!");}
         if (option.equals("1"))
    {
         System.out.print("input a message(msg1~msg5):");
           try{
            messageIn= stdin.readLine();
           }catch (Exception e)
         {System.err.pringln("cannot get a message!");}

        q1.Join(messageIn);
              messageTalk =q1.Leave();
              q2.Join(messageTalk);
              }

           if (option.equals("2"))
    {
        messageOut =q2.Leave();
               System.out.println(messageOut);
       }
    }
}


  public static void main(String args[]) {
    TwoActiveQueue myTAQ = new TwoActiveQueue();
     myTAQ.transfer();

   }
}
```

Appendix D

The *Queue* System in TCOZ

```
<?xml version="1.0" encoding="UTF-8"?> <ZML
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://nt-
  appn.comp.nus.edu.sg/fm/zml/zml.xsd" >
   <basicTypeDef>
      <name>MSG</name>
   </basicTypeDef>
   <axiomaticDef>
      <declaration>
         <variable>max</variable>
         <dataType>
            <type>N</type>
         </dataType>
      </declaration>
      <predicate>
         <expression>
            <varName>max</varName>
         </expression>
         <relationSym>gt</relationSym>
         <expression>
            <number>100</number>
         </expression>
      </predicate>
   </axiomaticDef>
   <schemaDef>
      <name>Queue</name>
      <declaration>
         <variable>items</variable>
         <dataType>
            <unarySym>seq</unarySym>
            <type>MSG</type>
         </dataType>
      </declaration>
      <predicate>
         <expression>
            <prefixExpr>#</prefixExpr>
            <expression>
               <varName>items</varName>
            </expression>
         </expression>
         <relationSym>leq</relationSym>
         <expression>
            <varName>max</varName>
         </expression>
      </predicate>
   </schemaDef>
   <schemaDef>
      <name>QueueInit</name>
      <incluList>Queue</incluList>
      <predicate>
         <expression>
            <varName>items</varName>
         </expression>
         <relationSym>=</relationSym>
         <expression>
            <left>lt</left>
            <right>gt</right>
         </expression>
      </predicate>
   </schemaDef>
   <schemaDef>
      <name>Add</name>
      <deltaList>Queue</deltaList>
      <declaration>
         <variable>itemIn</variable>
         <dataType>
            <type>MSG</type>
         </dataType>
      </declaration>
      <predicate>
         <expression>
```

```
            <expression>
               <varName>items</varName>
            </expression>
            <postfixExpr>'</postfixExpr>
         </expression>
         <relationSym>=</relationSym>
         <expression>
            <varName>items</varName>
         </expression>
         <relationSym>cat</relationSym>
         <expression>
            <left>lt</left>
            <expression>
               <varName>itemIn</varName>
            </expression>
            <right>gt</right>
         </expression>
      </predicate>
   </schemaDef>
   <schemaDef>
      <name>Delete</name>
      <deltaList>Queue</deltaList>
      <declaration>
         <variable>itemOut</variable>
         <dataType>
            <type>MSG</type>
         </dataType>
      </declaration>
      <predicate>
         <expression>
            <varName>items</varName>
         </expression>
         <relationSym>neq</relationSym>
         <expression>
            <left>lt</left>
            <right>gt</right>
         </expression>
      </predicate>
      <predicate>
         <expression>
            <varName>items</varName>
         </expression>
         <relationSym>=</relationSym>
         <expression>
            <left>lt</left>
            <expression>
               <varName>itemOut</varName>
            </expression>
            <right>gt</right>
         </expression>
         <relationSym>cat</relationSym>
         <expression>
            <expression>
               <varName>items</varName>
            </expression>
            <postfixExpr>'</postfixExpr>
         </expression>
      </predicate>
   </schemaDef>
   <classDef a="c1">
      <name>Queue</name>
      <state>
         <declaration>
            <variable>items</variable>
            <dataType>
               <unarySym>seq</unarySym>
               <type>MSG</type>
            </dataType>
         </declaration>
         <predicate>
            <expression>
               <prefixExpr>#</prefixExpr>
               <expression>
                  <varName>items</varName>
               </expression>
            </expression>
            <relationSym>leq</relationSym>
            <expression>
               <varName>max</varName>
            </expression>
         </predicate>
      </state>
      <initialState>
         <predicate>
            <expression>
               <varName>items</varName>
            </expression>
            <relationSym>=</relationSym>
            <expression>
               <left>lt</left>
               <right>gt</right>
            </expression>
         </predicate>
      </initialState>
      <operation a="o1">
         <name>Add</name>
         <deltaList>items</deltaList>
         <declaration>
            <variable>itemIn</variable>
            <dataType>
               <type>MSG</type>
            </dataType>
         </declaration>
         <predicate>
            <expression>
               <expression>
                  <varName>items</varName>
               </expression>
               <postfixExpr>'</postfixExpr>
            </expression>
            <relationSym>=</relationSym>
            <expression>
               <varName>items</varName>
            </expression>
            <relationSym>cat</relationSym>
            <expression>
               <left>lt</left>
               <expression>
                  <varName>itemIn</varName>
               </expression>
               <right>gt</right>
            </expression>
         </predicate>
      </operation>
      <operation a="o2">
         <name>Delete</name>
         <deltaList>items</deltaList>
         <declaration>
            <variable>itemOut</variable>
            <dataType>
               <type>MSG</type>
            </dataType>
         </declaration>
         <predicate>
            <expression>
```

```
            <varName>items</varName>
         </expression>
         <relationSym>neq</relationSym>
         <expression>
            <left>lt</left>
            <right>gt</right>
         </expression>
      </predicate>
      <predicate>
         <expression>
            <varName>items</varName>
         </expression>
         <relationSym>=</relationSym>
         <expression>
            <left>lt</left>
            <expression>
               <varName>itemOut</varName>
            </expression>
            <right>gt</right>
         </expression>
         <relationSym>cat</relationSym>
         <expression>
            <expression>
               <varName>items</varName>
            </expression>
            <postfixExpr>'</postfixExpr>
         </expression>
      </predicate>
   </operation>
</classDef>
<classDef a="c2">
   <name>TwoQueue</name>
   <state>
      <declaration>
         <variable>q1</variable>
         <dataType>
            <type>Queue</type>
         </dataType>
      </declaration>
      <declaration>
         <variable>q2</variable>
         <dataType>
            <type>Queue</type>
         </dataType>
      </declaration>
   </state>
   <operation a="o3">
      <name>Join</name>
      <operationExpression>
         <expression>
            <varName>q1</varName>
         </expression>
         <dot>.</dot>
         <name>Add</name>
      </operationExpression>
   </operation>
   <operation a="o4">
      <name>Leave</name>
      <operationExpression>
         <expression>
            <varName>q2</varName>
         </expression>
         <dot>.</dot>
         <name>Delete</name>
      </operationExpression>
   </operation>
   <operation a="o5">
      <name>Transfer</name>
      <operationExpression>
         <operationExpression>
            <expression>
               <varName>q1</varName>
            </expression>
            <dot>.</dot>
            <name>Delete</name>
         </operationExpression>
         <operationConnSym>parallel</operationConnSym>
         <operationExpression>
            <expression>
               <varName>q2</varName>
            </expression>
            <dot>.</dot>
            <name>Add</name>
         </operationExpression>
      </operationExpression>
   </operation>
</classDef>
<classDef a="c3">
   <name>ActiveQueue</name>
   <inheritedClass>
   <name>Queue</name>
</inheritedClass>
<state>
   <declaration>
      <variable>Ti</variable>
      <variable>Tj</variable>
      <dataType>
         <type>T</type>
      </dataType>
   </declaration>
   <declaration>
      <variable>in</variable>
      <variable>out</variable>
      <dataType>
         <type>Chan</type>
      </dataType>
   </declaration>
</state>
<operation a="o6">
   <name>Join</name>
   <processExpr>
      <guard>
         <declaration>
            <variable>item</variable>
            <dataType>
               <type>MSG</type>
            </dataType>
         </declaration>
         <predicate>
            <expression>
               <prefixExpr>#</prefixExpr>
               <expression>
                  <varName>items</varName>
               </expression>
            </expression>
            <relationSym>leq</relationSym>
            <expression>
               <varName>max</varName>
            </expression>
         </predicate>
      </guard>
      <processExpr>
         <event type="" var="">in</event>
         <then>then</then>
         <processExpr>
```

```
            <processExpr>
               <simpleProExp>Add</simpleProExp>
            </processExpr>
            <deadline>Tj</deadline>
         </processExpr>
      </processExpr>
   </processExpr>
</operation>
<operation a="o7">
   <name>Leave</name>
   <processExpr>
      <guard>
         <predicate>
            <expression>
               <varName>items</varName>
            </expression>
            <relationSym>neq</relationSym>
            <expression>
               <left>lt</left>
               <right>gt</right>
            </expression>
         </predicate>
      </guard>
      <processExpr>
         <event type="" var="">out</event>
         <then>then</then>
         <processExpr>
            <processExpr>
               <simpleProExp>Delete</simpleProExp>
            </processExpr>
            <deadline>Ti</deadline>
         </processExpr>
      </processExpr>
   </processExpr>
</operation>
<operation>
   <name>MAIN</name>
   <processExpr>
      <mu>Q</mu>
      <processExpr>
         <processExpr>
            <processExpr1>
               <simpleProExp>Join</simpleProExp>
            </processExpr1>
            <proConnSym>externalChoice</proConnSym>
```

```
                  <processExpr2>
                     <simpleProExp>Leave</simpleProExp>
                  </processExpr2>
               </processExpr>
               <proConnSym>composition</proConnSym>
               <processExpr>
                  <simpleProExp>Q</simpleProExp>
               </processExpr>
            </processExpr>
         </processExpr>
      </operation>
   </classDef>
   <classDef a="c4">
      <name>TwoActiveQueue</name>
      <state>
         <declaration1>
            <variable>q1</variable>
            <dataType>
               <type>ActiveQueue</type>
               <renameList>talk/out</renameList>
            </dataType>
         </declaration1>
         <declaration2>
            <variable>q2</variable>
            <dataType>
               <type>ActiveQueue</type>
               <renameList>talk/in</renameList>
            </dataType>
         </declaration2>
      </state>
      <operation>
         <name>MAIN</name>
         <processExpr>
            <activeObject1>q1</activeObject1>
            <channelName>talk</channelName>
            <activeObject2>q2</activeObject2>
         </processExpr>
      </operation>
   </classDef>
</ZML>
```