

## INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning  
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA  
800-521-0600

UMI<sup>®</sup>



**Policy Capturing And Two Level Specifications Of  
Policies For Human And Software Processing**

**Xin He**

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfilment of the Requirements

for the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

March 2002

©Xin He, 2002



National Library  
of Canada

Acquisitions and  
Bibliographic Services

395 Wellington Street  
Ottawa ON K1A 0N4  
Canada

Bibliothèque nationale  
du Canada

Acquisitions et  
services bibliographiques

395, rue Wellington  
Ottawa ON K1A 0N4  
Canada

*Your file Votre référence*

*Our file Notre référence*

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-68467-9

**Canada**

## **Abstract**

Policy Capturing And Two Level Specifications Of Policies For Human And Software  
Processing

Xin He

In recent few years, the use of “policies” in the management of the behavior of large-scale, dynamic or distributed systems is gaining importance. In the policy research area, how to capture policies is the first problem scientists need to solve. Policies can be specified in three ways: a Policy Definition Language (PDL), a formal logic-based policy representation language or an informal natural language. Generally, an editor and a policy management tool are used to input these specifications. Because both the PDLs and the formal logic-based policy representation languages are very complex and more suitable for machine communication instead of human communication, their usages are limited to a small group of professional people.

In this thesis, two levels of policy specifications are introduced. A Restricted Natural Language (RNL), as the high-level policy specification, is used for human-computer interactions. Compared to the PDLs and the formal logic-based policy representation languages, the RNL is both easy to use and easy to learn. It enables a wider group of end users, especially the novice users, to participate in stating the policies. A logic-based formal representation, as the low-level policy specification, is then used for further machine-oriented processing in policy-based systems.

A methodology of how to construct such a RNL is also presented and the methodology is applied to two different cases. In the implementation part of the thesis, a RNL user interface (RNLI) named the Virtual Thesis Office (VTO) System is also developed and presented.

## Acknowledgements

I would like to take this opportunity to express my sincere thanks to my thesis supervisor **Dr. Thiruvengadam Radhakrishnan** for providing guidance, advice, and financial support throughout my studies.

I wish to thank **Dr. Franz Kurfess** for creating interest in me for graduate studies.

The financial support provided by Nortel Networks, Ottawa through a research grant given to **Dr. T. Radhakrishnan** gratefully acknowledged.

I would also like to thank **Angus Graham** and **Venkatachalam Kanthimathinathan**, who have unselfishly offered their comments and supported test cases on this research.

## Table of Contents

<b>Chapter</b>	
<b>1</b>	<b>Introduction</b> <span style="float: right;">1</span>
<b>2</b>	<b>Policy Overview</b> <span style="float: right;">5</span>
2.1	Policy Definition and Policy Hierarchy . . . . . 6
2.2	Policy Types . . . . . 7
2.3	Policy-Based System Architecture . . . . . 8
2.4	Policy Representation . . . . . 11
2.4.1	Analysis of Policy-Description Language Based Approaches . . . . . 11
2.4.2	Analysis of Formal Logic-Based Approaches . . . . . 12
2.4.3	Case Study . . . . . 13
2.5	Policy Capturing . . . . . 20
<b>3</b>	<b>Natural Language Interface</b> <span style="float: right;">22</span>
3.1	Implementing a Natural Language Interface . . . . . 22
3.2	A Model of Natural Language Processing (NLP) . . . . . 23
3.2.1	Lexical Analysis . . . . . 25
3.2.2	Syntactic Analysis . . . . . 25
3.2.3	Semantic Analysis . . . . . 31
3.3	Habitability . . . . . 33

<b>4</b>	<b>A Restricted Natural Language for Policy Capturing</b>	<b>34</b>
4.1	User-Centered Design for Policy Capturing . . . . .	35
4.2	Level I - The Restricted Natural Language . . . . .	39
4.3	Level II - A Formal Logic-based Language (FR) . . . . .	41
4.4	A Methodology to Construct a Restricted Natural Language for Policy Capturing . . . . .	42
4.5	The Expandability of the RNL . . . . .	43
<b>5</b>	<b>Case Study</b>	<b>45</b>
5.1	Virtual Thesis Office (VTO) . . . . .	45
5.1.1	Step1: Define the Problem Domain . . . . .	45
5.1.2	Step2: Create the Object-Oriented Model . . . . .	56
5.1.3	Step3: Construct a Vocabulary . . . . .	60
5.1.4	Step4: Construct the Syntax . . . . .	62
5.1.5	Step5: Manually Verify the RNL's Habitability . . . . .	67
5.1.6	Step6: Develop a Formal Logic-based Policy Specification . . . . .	68
5.1.7	Step7: Develop a Mapping from RNL to the FR . . . . .	75
5.2	Virtual Call Center (VCC) . . . . .	81
5.2.1	Step1: Define the Problem Domain . . . . .	82
5.2.2	Step2: Create the Object-Oriented Model . . . . .	83
5.2.3	Step3: Construct a Vocabulary . . . . .	85
5.2.4	Step4: Construct the Syntax . . . . .	85
5.2.5	Step5: Manually Verify the RNL's Habitability . . . . .	85
5.2.6	Step6: Develop a Formal Logic-based Policy Specification . . . . .	85
5.2.7	Step7: Develop a Mapping from RNL to the FR . . . . .	87
5.3	Summary . . . . .	88
<b>6</b>	<b>VTO System Design and Implementation</b>	<b>90</b>



6.1	Implementation Environment . . . . .	90
6.2	Software Architecture . . . . .	90
6.3	Policy Management Tool . . . . .	91
6.3.1	The Time Sequence of the Policies . . . . .	91
6.4	RNL Processor . . . . .	96
6.4.1	Lexical Analyzer . . . . .	96
6.4.2	Parser . . . . .	100
6.4.3	Semantic Analyzer . . . . .	101
6.4.4	Ambiguity Problem . . . . .	103
6.5	The Verification of the Software . . . . .	105
<b>7</b>	<b>Conclusion And Future Work</b>	<b>106</b>
7.1	Conclusions . . . . .	106
7.2	Future Work . . . . .	108
	<b>Bibliography</b>	<b>110</b>
	<b>Appendix</b>	
<b>A</b>	<b>The regualtions for the examinations of the master's theses</b>	<b>114</b>
<b>B</b>	<b>Class Definition</b>	<b>116</b>
<b>C</b>	<b>A Complete List of Re-written Policies Using VTO System's RNL</b>	<b>121</b>

## List of Tables

### Table

2.1	Goal-oriented policy . . . . .	16
2.2	A Class of Objects/Agents . . . . .	18
2.3	Class <i>Mission</i> . . . . .	19
3.1	An Example of a Lexicon . . . . .	25
3.2	Fillmore's Case System . . . . .	32
5.1	A List of Words with Fixed Unique Meanings . . . . .	62
5.2	The Hidden Template for the policy " <i>Each examiner must sign the evaluation form</i> " . . . . .	76
5.3	The Mapping Relationship between the HT and RNL's Syntax . . . . .	78
6.1	An Example of a HT . . . . .	102

## List of Figures

### Figure

2.1 Masullo and Calo's Policy Hierarchy . . . . .	7
2.2 A Policy-Based System Architecture . . . . .	9
2.3 Authorization Policy Syntax . . . . .	14
2.4 An Example of Authorization Policies . . . . .	14
2.5 Obligation Policy Syntax . . . . .	15
2.6 An Example of Obligation Policies . . . . .	16
2.7 An Example of Operational Level Policies . . . . .	17
3.1 A Model of an Natural Language Processing System . . . . .	24
3.2 An Example of Context-Free Grammar . . . . .	27
3.3 Parse Tree of the input " <i>Gloria saw the man.</i> " . . . . .	27
3.4 An Example of Recursive Transition Network . . . . .	29
3.5 Generating an LR Parser . . . . .	30
5.1 The Use Case Map for the VTO system . . . . .	47
5.2 The Use Case Map for the VTO system - Continue . . . . .	48
5.3 Class Diagram for the VTO Model and its generalization relationships . . . . .	57
5.4 ECm Class . . . . .	60
5.5 The Verbs Listed in the Vocabulary . . . . .	63
5.6 Eight Grammar Rules for VTO System's RNL . . . . .	64
5.7 Some Examples for VTO System's RNL Grammar . . . . .	65

5.8	The Sementic Meaning of “before” in the Vocabulary . . . . .	79
5.9	The Semantic Meaning of “request a postponement to” in the Vocabulary .	80
5.10	The Semantic Meaning of “request a postponement to” in the Vocabulary .	81
5.11	The Use Case Map for the VCC system . . . . .	82
5.12	Class Diagram for the VCC Model and the Verbs Listed in the Vocabulary	84
5.13	The Vocabulary for the VCC system . . . . .	86
5.14	The Grammar Used in VCC system . . . . .	86
6.1	The VTO system’s Architecture . . . . .	92
6.2	The VTO System’s Policy Management Tool . . . . .	93
6.3	The Time Scope of Policies . . . . .	94
6.4	The Process of Lexical Analysis . . . . .	99
6.5	A Parse Tree Generated from the Parser . . . . .	100

## Chapter 1

### Introduction

The use of “policies” in the management of the behavior of large-scale, dynamic or distributed systems is gaining importance. Many researchers and organizations have proposed generic architectures and frameworks for policy-based systems. For example, in [30], Lupu and Sloman provide a role-based policy management framework to manage distributed systems. IETF (Internet Engineering Task Force) has also proposed several standards such as Policy Core Information Model (CIM), Security Policy Specification Language (SPSL), Quality of Service (QoS) policies, and Lightweight Directory Access Protocol (LDAP) regarding the policy-based system management [16].

In the policy research area, how to capture policies is the first problem scientists need to solve. A policy can be viewed as a set of actions, which are triggered under certain conditions and act on the target objects. Policies are communicated between humans in a natural language form. This form can be very abstract, may contain ambiguities and conflicts, and it is very difficult to be understood by the computers. To make them understandable by machines, we must first capture the policies and translate them into a formal form. In a formal form the policies can be analyzed for the purposes of disambiguation, conflict resolution or further refinement.

Currently, the typical way to capture the policies directly use formal languages. The end users need to use a Policy Definition Language (PDL) or a formal logic-based policy representation ([25], [45], [12], [29] and [11]) to write down each policy by hand first, then input them into the policy-based system using a policy management tool.

The PDL is used to represent policies at the “operational level”. It may contain complex syntax in order to achieve the coverage of different policies. People who want to use it should first remember the syntax and terminology that represents the subjects, objects or actions etc. in the problem domain. This requirement is too severe, for most novice end users and seldom people have this skill without suitable training. Because of this, the use of PDL for policy capturing is restricted to a small group of well-trained professional policy engineers.

The formal logic-based policy representation approach, on the other hand, has the ability to represent both the “operational level” policies and the “abstract level” policies. However, it has the same problem the PDL has. First, it involves some logical and mathematical symbols that look difficult to novice users. Secondly, people who want to use it still need to remember lots of terminology representing the subjects, objects and actions etc. inside the problem domain. These difficulties limit the use of the formal logic-based approach into an even smaller group of professional people.

To overcome the disadvantages of the way people currently capture policies, we propose two levels of policy specifications and introduce a methodology to construct a Restricted Natural Language and the corresponding interface (RNLI) for policy capturing in this thesis.

The two levels of policy specifications include a Restricted Natural Language (RNL) for end users’ input and a Formal Representation (FR) for the policy specification.

A policy stated in the RNL can be seen as a higher level of policy specification in the policy hierarchy described in Section 2.1. A RNL is a subset of the natural language and is restricted in the lexicon, the syntax and the semantics. Since a RNL is very much like a natural language, using it to input policies into the policy-based system (policy capturing) is much easier than using PDLs and other formal languages.

The FR is seen as a lower level of policy specification in the policy hierarchy (see Section 2.1). The formal form we adopt for the FR in this thesis is a combination of

an object-oriented model and some logic operators. The FR is used to describe subjects, objects, actions and the trigger conditions in the policies. Because of its preciseness, representing policies in the FR is suitable for automatic processing such as automatic policy refinement, static policy conflict analysis and policy enforcement.

A Restricted Natural Language Interface (RNLI) is an interface that allows users to use the Restricted Natural Language (RNL) to input their policies and translate the policies in RNL format to FR format, then store them into the policy-based systems. Compared to the current policy capturing ways which use PDLs or formal logic-based languages, the RNLI is obviously easier to use and easier to learn for most end users especially the novice users. We hope such kind of interfaces will widen the group of people who participate in policy capturing.

The RNLI is made up of five components:

- (1) a concept dictionary or a lexicon for the specific domain.
- (2) a simple grammar to construct the lexicon into policy sentences.
- (3) an object-oriented model for representing the knowledge expressed in the lexicon.
- (4) a formal form of the policy representation for the behavior of the specific domain.  
and
- (5) a parsing technique to map the RNL into the formal form of the policy representation.

The methodology for constructing a RNL and the corresponding RNLI for policy capturing proposed in this thesis is a general method for constructing different RNLs and RNLIs in different problem domains. The Virtual Thesis Office (VTO) and the Virtual Call Center (VCC) presented in this thesis are two examples to demonstrate how to construct a RNL and the corresponding RNLI for a specific problem domain using our methodology.

The contents of the thesis are organized as outlined below:

Chapter 2 examines the issues involved in the definition, representation and capturing of the policies. Several approaches of how to represent policies as found in the literature are described.

Chapter 3 introduces the main aspects of the development of a natural language interface (NLI).

In Chapter 4, two levels of policy specifications - a Restricted Natural Language (RNL) and a formal logic-based language (FR) are proposed to represent policies. A methodology of how to construct such a RNL and the corresponding RNLI for policy capturing are also introduced.

The Virtual Thesis Office (VTO) and the Virtual Call Center (VCC), as two examples used for demonstrating the above methodology are described in Chapter 5. The detailed system design and implementation of the VTO system is given in Chapter 6.

Finally in Chapter 7 the conclusions of this thesis are summarized and the scope for further work is suggested.



## Chapter 2

### Policy Overview

The use of policies in automatic management has increased rapidly in distributed system management, network management, security and privacy management, as well as in the control and management of the specialized needs of the end users. Several researchers and organizations have proposed generic architectures and frameworks for policy-based systems. In [30], Lupu and Sloman provide a role-based policy management framework to manage distributed systems. Policies can be specified by different groups of people such as administrators, or managers, end users, service providers etc. Policies in a system can be statically specified at “compile time” or dynamically modified at “run time”. IETF (Internet Engineering Task Force) has also proposed several standards such as Policy Core Information Model (CIM), Security Policy Specification Language (SPSL), Quality of Service (QoS) policies, and Lightweight Directory Access Protocol (LDAP) regarding the policy-based system management [16].

The reason policy-based systems are so attractive is because of their significant features. Instead of being coded into the software components, policies can be stored into their own policy repository and can be interpreted, triggered and changed dynamically. Such a feature increases the software components' flexibility and re-usability. Furthermore, when using policy-based systems, human managers can keep focus more on the business model than the technical aspects of the systems because policies can be specified in an abstract level, which is much closer to the system's business models.

## 2.1 Policy Definition and Policy Hierarchy

In Merriam-Webster dictionary, the word policy is defined as “a definite course or method of action selected from among alternatives and in light of given conditions to guide and determine present and future decisions” or “a high-level overall plan embracing the general goals and acceptable procedures especially of a governmental body”. Depending on the situation, the meaning of policies may be difficult. In an organization, policies could be the plans to achieve its objectives [34]. Here the organization’s leaders define the policies. These policies are usually written in a natural language and are seen as the directions to be followed in achieving the goals or to derive low level policies to achieve these goals. The policies are abstract and cannot be executed directly without detailing the contents into operational actions. On the contrary, a policy in a specific network management system could be clearly defined as an action which takes effect on the specific object in the system and is triggered by a specific condition. The definer of this kind of policies could be the system administrator who is directly working on the system. The executor of the policies is the machine itself. The results of the execution of this kind of policies are easy to measure - they cause the system’s state transferred from one to another. Policies can be specified at different levels of abstraction. Policy Hierarchy explains the relationship among these different levels’ of policies.

Policies may range from high level abstract non-technical policies to low level technical policies. As shown in Table 2.1, Masullo and Calo [33] group policies into six distinct levels. In [45], Wies separate them into four classes. More technically oriented, Koch [25] put them into three levels.

Dividing high level policies into low level ones is known as “stepwise refinement”, which topic is studied also in the field of software engineering. Policy refinement can be done by refining the goals, partitioning the targets or delegating the responsibility to other managers.

In [33]’s hierarchy, societal, directional and organization policies focus more on the

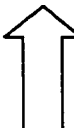
	<i>Masullo &amp; Callo(1993)</i>	<i>Wies(1995)</i>	<i>Koch(1996)</i>
Degree of Abstraction Level 	Societal Policy	Corporate/High-Level Policy	
	Directional Policy		
	Organizational Policy	Task Oriented Policy	
	Functional Policy	Functional Policy	Requirement Level Policy
	Process Policy		Goal-oriented Level Policy
	Procedural Policy(Rule)	Low-Level Policy	Operational Level Policy

Figure 2.1: Masullo and Calo's Policy Hierarchy

business economic aspects and should only be constructed by humans using their talent and intuition. Therefore, the discussion of these kinds of policies is out of our thesis's scope and they are not addressed here.

The policies described in this thesis is in between the functional level and process level in Masullo & Calo's hierarchy. In this level, a policy's subjects and targets are specific entities which have clear definitions in a system management environment: the behaviors of a policy can be mapped to a specific action or a set of actions together with the authorization/obligation information: the expression of a policy can be formalized in order to process further refinement steps: and the result of the execution of a policy can be simply specified by a system changing its status from one to another.

**Definition:** A policy defines a set of actions, which are triggered under certain conditions, acting on the target objects. When actions defined in the policy are triggered, they cause the target system environment to change from one state to another.

## 2.2 Policy Types

According to Sloman's model [30], authorization and obligation policies are two main types of policies in the policy area.

**Authorization policies** specify what action the subjects are authorized or forbidden to invoke on the target objects. Examples of this type of policies include:

**Example1:** *Only root has the authority to modify /etc/default/cron file.*

**Example2:** *Any member of the University may attend the candidate's thesis defence.*

When an authorization policy is specified, whether or not to do the actions defined in the policy depends on the policy executor's intention at the run time. Therefore, the result of an authorization policy is real dynamic and cannot be predicated until the events actually happen.

**Obligation policies** specify the actions that subjects must or must not perform on the target objects.

**Example3:** *Each member of the examining committee must submit a written report on the thesis to the chair before the thesis defence begins.*

**Example4:** *The chair must return the examination report to the thesis officer.*

An obligation policy describes the policy executor's duties. When the conditions of triggering the policy is satisfied, the actions defined in the policy must be performed and should lead the system status to a predictable result. If any unexpected result happens, the system will be in an error state.

### 2.3 Policy-Based System Architecture

A policy-based system usually contains the following components.

**Policy Management Tool** accepts high level or natural language statement of policies from the Policy Capturing Interface. It first translates the policies into formal logical form, then refines the high level policies into the operational level policies. Finally, it checks and resolves the static conflicts.

**Policy Capturing Interface** provides an interface for policy providers to input policies into the Policy Management Tool. It is ideal to use a natural language to exchange information between the system and the end users just like the humans communicate.

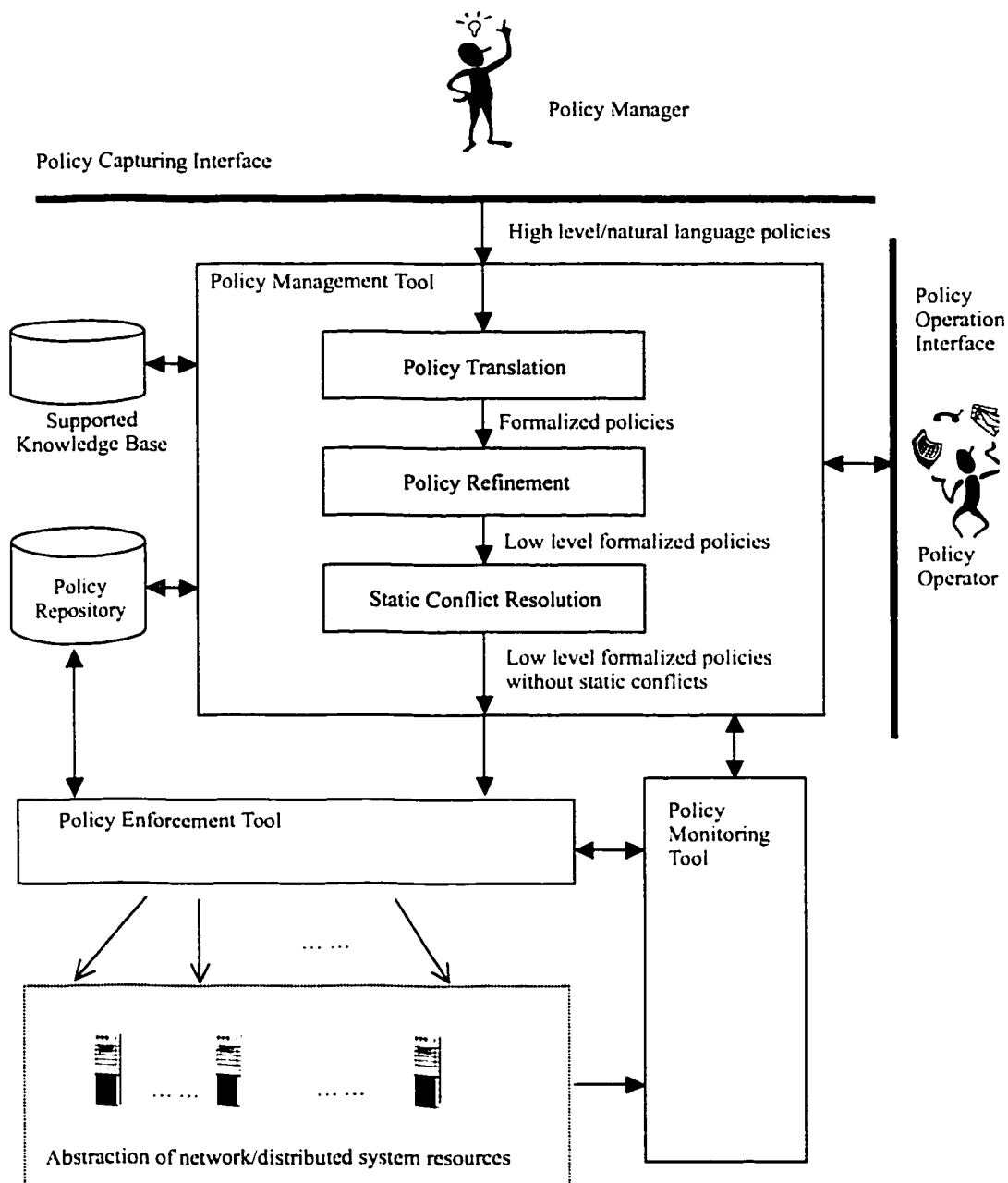


Figure 2.2: A Policy-Based System Architecture

**Policy Operation Interface** provides an interface for policy operators to enable or disable policies in the policy repository. Operators don't have the right to define policies. Their responsibility is to select the existing policy and send them to the Policy Enforcement Tool. After the policy system is well defined or the policy repository is created, the whole system will run automatically. This interface only provides an auxiliary way for human operators to fine-tune the automatic systems.

**Policy Enforcement Tool** receives the policies from the Policy Management Tool and enforce them to the appropriate system objects.

**Policy Monitoring Tool** monitors the policy enforcement process. It collects the system states and sends them back to the system objects as the conditions to trigger policies. This is also the place to detect and resolve dynamic policy conflicts.

**Supported Knowledge Base** is the place to store lexicon, domain knowledge, refinement meta-rules and conflict resolving meta-rules for the Policy Management Tool, which performs translation, refinement and conflict resolution.

**Policy Repository** is the place to store all the policies.

In the Policy Management Tool, the following components are essential.

- (1) **The Component for Policy Translation:** An abstract level policy which is captured from the end user may be in a natural language format. Thus, we need a tool to translate the natural language policy into some formalized form to
  - (a) discard those "error" policies which contain ill syntax or semantics.
  - (b) reduce the ambiguity
  - (c) resolve the incompleteness

A formal-logic based policy representation is usually used for the formal form.

- (2) **The Component for Policy Refinement:** Abstract policies cannot be directly implemented. A stepwise refinement is needed to detail the abstract policies into machine-understandable operational level policies.
- (3) **The Component for Static Conflict Resolution:** Conflicts may arise in the policy refinement process. Thus, a static conflict check should be done after the refinement process is complete.

## 2.4 Policy Representation

Policy representation, as one of the fields in policy research, has the goal of establishing a formal form to represent policies, which makes automated analysis, verification and refinement of policies possible.

In recent years, scientists have proposed several approaches for policy representation ([25], [45], [12], [29] and [11]). Generally, they can be divided into two groups: description-language based approaches and formal-logic based approaches.

### 2.4.1 Analysis of Policy-Description Language Based Approaches

The Policy-Description Language (PDL) based approaches have some common characteristics:

- They all use Backus Naur Form as the format of the languages.
- They contain fixed formats for each kind of policies. Once a new kind of policy is created, a new set of syntax should be developed.
- For each syntax, they predefine a set of columns to contain every aspects of a policy. Therefore, once a policy is described in a PDL, it is quite easy for a machine to interpret it into a machine-understandable language.
- Because of the variety of the policies, the syntax of PDLs could be very complex in order to achieve the coverage for each kind of policy. This makes the languages

difficult to learn and without memorizing the basic parts of the syntax, nobody can use the languages. This aspect limits the languages' usability to only a few professional people who can become familiar with the them.

- These languages are very good in describing operational level policies because of the explicitness of operational level policies. However, it is inappropriate to use them to express abstract level policies because of the following reasons.
  - (1) Expressing abstract level policies makes the language lose its “easy-to-interpret” advantage since the semantics of abstract level policies may not be explicit. To reduce the ambiguity and dismiss the indistinctness and to do the stepwise refinement, the system needs a domain-dependable knowledge base. Such a refinement is a hard process and may not be 100% correct.
  - (2) In the abstract level policies, the triggers or constraints may not be clearly declared. This makes it hard to incorporate an abstract level policy into a clearly predefined PDL.

Approaches of policy description languages includes [12], [29], [25] and [41]. We will use Ponder language [12] as an example to briefly introduce PDLs in Section 2.4.3.1.

#### **2.4.2 Analysis of Formal Logic-Based Approaches**

Formal-logic based approaches usually combine an object model to represent the entities in the policy domain and one or more logic models to express the logical or time elements and the authorization/obligation concepts. The Cuppens' deontic logic based approach has been used to formalize security policies [11], which will be explained in Section 2.4.3.3. The key features of such approaches are listed as follows.

- Abstract level policies are represented in the same format as the operational level policies. It is convenient to represent an abstract level policy by introducing abstract level objects into the representation and use logics to represent the relation-



ships among abstract objects. The abstract level objects are combined with some attributes and methods. These attributes and methods are also in an abstract level. The relationships among abstract objects and the real entities can be expressed in an object-oriented data model. In this way the abstract level policies are expressed in a formal manner. Thus, mathematical methods can be used for further policy refinements. The underlying object-oriented model becomes a knowledge resource for going through the refinement process.

- When mathematical logic is used for the formal representation, it is possible to create proved correct and complete refinement patterns [13] that can be used to refine the policies. Thus, high level policies can be correctly refined into elementary level policies.

### 2.4.3 Case Study

#### 2.4.3.1 Ponder Policy Specification Language

Ponder language [12] is an object-oriented language for specifying security and management policy for distributed object systems. It provides a complex syntax for “access control policies” and “obligation policies”. It also provides the syntax to organize policies into groups.

Access control policies are those policies which are used to limit the activity of legitimate users who have been successfully authenticated. Ponder supports access control by providing **authorization**, **delegation**, **information**, **filtering** and **refrain** policies.

Authorization policies define what activities a member of the subject domain can perform on the set of objects in the target domain. An example of an authorization policy syntax is shown in Figure 2.3

Based on the above syntax, we can express the policy “*Members of the NetworkAdmin domain are authorized to load, remove, enable or disable objects of type PolicyT in the Nregion/switches domain.*” in the format shown in Figure 2.4.

```

inst ( auth+ / auth- ) policyName {
  subject [<type>] domain-Scope-Expression ;
  target  [<type>] domain-Scope-Expression ;
  action          action-list ;
  [ when          constraint-Expression ; ]
}

```

Figure 2.3: Authorization Policy Syntax

In natural language the policy is:

*“Members of the NetworkAdmin domain are authorized to load, remove, enable or disable objects of type PolicyT in the region/switches domain.”*

In Ponder language the policy is:

```

inst auth+ switchPolicyOps {
  subject          /NetworkAdmin;
  target <PolicyT> /NRegion/switches;
  action          load(), remove(), enable(), disable() ;
}

```

Figure 2.4: An Example of Authorization Policies

Obligation policies specify the actions that must be performed by managers within the system when certain events occur. An example of obligation policy syntax is shown in Figure 2.5.

```

inst ( oblig )           policyName {
  on                    event-specification ;
  subject [ <type> ]    domain-Scope-Expression ;
  [ target [ <type> ]  domain-Scope-Expression ; ]
  do                   obligation-action-list ;
  [ catch              exception-specification ;
  [ when               constraint-Expression ; ]
}

```

Figure 2.5: Obligation Policy Syntax

Figure 2.6 is an example about how to use the obligation policy syntax.

This policy is triggered by 3 consecutive login-failure events with the same *userid*. The NRegion security administrator (*SecAdmin*) disables the user with *userid* in the */NRegion/users* domain and then logs the failed *userid* by means of a local operation performed in the *SecAdmin* object.

#### 2.4.3.2 Koch's Policy Definition Language

In [25], Koch proposed a three level policy hierarchy and used a policy definition language to express the policies on the operational level. At the requirement level the desired behavior in an abstract level and is specified in natural language, as in the following example.

Garbage: "The **administrator** should immediately remove all **garbage-files** if more than 80% of the **available disk-space** on a **home partition** is used."

The goal-oriented level is an intermediate level at which the roles of the objects involved are defined in terms of constraints and actions. For example, Table 2.1 shows a second level refinement of the above (with mandatory attributes in boldface).

In natural language the policy is:

*“If the user has failed to login the system using the same userid for three times, then the Nregion security administrator disables the user with userid in the /Nregion/users domain and then logs the failed userid into a log file.”*

In Ponder language the policy is:

```
inst oblig loginFailure {
  on          3*loginfail(userid) ;
  subject     s = /NRegion/SecAdmin ;
  target <UserT> t = /NRegion/users ^ {userid} ;
  do          t.disable() -> s.log(userid) ;
}
```

Figure 2.6: An Example of Obligation Policies

<b>Name</b>	Goal03
Ancestor	Goal02
Descendant	disk80
<b>Subject</b>	root
<b>Target Object</b>	none
<b>Action</b>	removeGarbage()
<b>Constraint</b>	none
<b>Event</b>	disk-usage > 80%
<b>Modality</b>	Obligation
Status	refined
Author	Thomas Koch

Table 2.1: Goal-oriented policy

The operational level policy specification (shown in Figure 2.7) employs the policy description language (PDL) to express the policy in a form which can be translated into executable rules conveniently.

```

policy disk80 type obligation for dvt/root {
    for every ExtendedFileSystem in filesys/home
    actiondisk_use.use80 {
        removeGarbage()
    } nosuccess ( dvt/status.filesys == ALARM )
}

```

Figure 2.7: An Example of Operational Level Policies

### 2.4.3.3 Cuppens' Deontic Logic Approach

In [11], Cuppens introduced a deontic logic-based policy representation approach to formalize security policies. According to his analysis, security policies may contain the following concepts:

- objects, events, actions and agents
- most of classical deontic concepts such as obligation, permission, and prohibition
- concepts of responsibility and delegation
- the temporal structures

He constructed a deontic language to represent all the above concepts. He used an object-oriented representation to express objects as shown in Table 2.2.

Events are represented as objects. Three temporal concepts *before*, *during* and *after* are defined as follows.

$\forall e, \text{Before}(e) \leftrightarrow$

$$\exists t_1, t_2, \text{Date}(e) = \langle t_1, t_2 \rangle \wedge \text{Current\_Date} < t_1$$

<b>Class:</b>	Classified_Document
<b>Inherits:</b>	Document
<b>Attributes:</b>	Classification: Level Consignment_Note: Note
<b>Methods:</b>	Change_Classification: Classified_Document → Level Establish_Consignment_Note: Classified_Document → Note

Table 2.2: A Class of Objects/Agents

$$\forall e, \textit{During}(e) \leftrightarrow$$

$$\exists t_1, t_2, \textit{Date}(e) = \langle t_1, t_2 \rangle \wedge t_1 \leq \textit{Current\_Date} \leq t_2$$

$$\forall e, \textit{After}(e) \leftrightarrow$$

$$\exists t_1, t_2, \textit{Date}(e) = \langle t_1, t_2 \rangle \wedge t_2 < \textit{Current\_Date}$$

Actions are defined in the same way as the methods are defined in an object-oriented environment. In addition, three corresponding binary predicates are clarified: if  $a$  is an agent and if  $\alpha$  is an action, then

- $\textit{Before\_Exec}(a, \alpha)$  is true if the value of  $\textit{Current\_Date}$  is previous to the time of the execution of  $\alpha$  by  $a$ .
- $\textit{During\_Exec}(a, \alpha)$  is true during the execution of  $\alpha$  by  $a$ .
- $\textit{Exec}(a, \alpha)$  is true after  $a$  has executed  $\alpha$ .  $\textit{Exec}$  is actually an abbreviation for  $\textit{After\_Exec}$ .

Deontic concepts are expressed by classical deontic modalities.

**Op** is to be read  $p$  is obligatory:

**Fp** is to be read  $p$  is forbidden:

**Pp** is to be read  $p$  is permitted.

<b>Class:</b>	Mission
<b>Attributes:</b>	Responsible: Agent Controller: Agent Obligations: set of Action
<b>Methods:</b>	Designate_Responsible: Mission $\rightarrow$ Agent

Table 2.3: Class *Mission*

In order to formalize the organizational concepts of responsibility and delegation, the special object class *Mission* is defined with the structure shown in Table 2.3:

Therefore, if  $m$  is a mission and  $a$  and  $b$  are two agents, then  $Responsible(m) = a$  means that  $a$  is responsible for the mission  $m$  and  $Controller(m) = b$  means that  $b$  is the agent who is in charge of controlling the execution of  $a$ 's mission. If  $a$  is an agent and  $\alpha$  is an action, then  $Delegate(a, \alpha)$  corresponds to the action of delegating the obligation of executing the action  $\alpha$  to the agent  $a$ .

An example policy is shown as follows using the above definitions.

**Policy:** The transmitter of a classified document is **obliged** to update the document classification as soon as it is possible, i.e. immediately after the transmitter evaluates that the document classification is obsolete.

**Rule R1:**

Classified\_Document(d)  
 $\wedge$  Transmitter(d)=t  
 $\wedge$  Res\_Exec(t, Evaluate\_Classification(d), level)  
 $\wedge$  Classification(d)  $\neq$  level  
 $\rightarrow$  **O** Exec(t, Change\_Classification(d))

One advantage of using deontic logic to express policies is a policy with complex structures can be expressed directly, while it is hard to do using PDL. The following is such an example. We can see that Cuppen's logic based approach is suitable to express such a

policy.

**Policy:** If the organizer of a meeting does not incinerate all the preparatory documents of the meeting when it is finished, then he is obliged to bring it about that these documents are kept in a safe.

**Rule R12:**

$$\begin{aligned}
 & \text{MEETING}(m) \\
 & \wedge \text{Organizer}(m) = o \\
 & \wedge \text{After}(m) \\
 & \wedge d \in \text{Preparatory\_Documents}(m) \\
 & \wedge \neg \text{Exec}(o, \text{Incinerate}(d)) \\
 & \quad \rightarrow \mathbf{O} \text{Do}(o, \exists s. (\text{Safe}(s) \wedge d \in \text{Content}(s)))
 \end{aligned}$$

## 2.5 Policy Capturing

As shown in Figure 2.2, the goal of policy capturing is to provide an efficient and convenient way for the users to transfer their knowledge about policies of the system to the policy management software systems.

Currently, many research efforts have been devoted to the policy representation area. Several policy description languages and formal-logic based representation approaches have been proposed as we saw in the previous sections. These approaches, however, are complex to the end-users. Consider the following problem: How can we add a shell outside the policy description languages or the formal-logic based representations to make the interface easier to communicate with end-users, and maintain the acceptable response time at the same time?

In an organization, policies are often recorded in written natural language. Short and transient policies may also be expressed orally. It would be convenient if policy management systems could capture such policies directly in suitable natural language form. In this thesis,



we propose a Restricted Natural Language (RNL) for policy capturing. The RNL is a subset of the natural language with restrictions in lexicon, syntax and semantics. The details are explained in Chapter 4 and 5.

## Chapter 3

### Natural Language Interface

The main goal of most natural language interfaces (NLI) is to provide users interfaces that are easy to learn and easy to use. In this chapter, we provide an overview and a brief introduction to NLI. Complete surveys and extensive descriptions of NLI are abundant in the literatures [10] and [3]. Natural language research attempts to develop a technique to make the NLI “understand” the semantics or the meaning of the user’s input, and provides intelligent response. When combined suitably with the speech recognizers, the natural language interface will minimize the training requirement, release the users’ burden of learning numerous artificial command languages, and support hands-free or mobile usage.

#### 3.1 Implementing a Natural Language Interface

Despite the intuitive appeal of a natural language interface, it has been argued that a natural language like English has too many ambiguities to be useful for communicating with computers. The research in [36] indicates that a restricted natural language interface is better suited for novice users when it is used for solving only simple problems. When problems get more complex, the ambiguity of the restricted natural language will pose a challenge in understanding the user’s command.

The ambiguities are found at every linguistic level in natural language specifications: lexical, syntactic, semantic and pragmatic. Examples of different kinds of ambiguities are shown as follows ([20]).

**Lexical ambiguity:** *The astronomer married the star.*

Does the word “*star*” mean a planet or a Hollywood actor/actress?

**Syntactic ambiguity:** *Time flies like an arrow.*

What is the verb in this sentence, “*flies*” or “*like*”?

**Semantic ambiguity:** *The professor sent the student to see the dean, because he wanted to see him.*

Who is “*he*”/“*him*”, the “*student*” or the “*textitdean*”?

**Pragmatic ambiguity:** *I saw the Grand Canyon flying to New York.*

What is “*Grand Canyon*”?

There is no perfect way developed yet to solve the ambiguity problem in natural language systems. Many efforts ([26], [17] and [10]) are undertaken by researchers to solve or avoid such problems. Examples of these efforts can be grouped into the following categories:

- (1) Use rule-based or probabilistic disambiguator to solve ambiguity.
- (2) Use dialogue to interactively assist in the disambiguation of the input.
- (3) Restrict the lexicon and the syntactic structures into a certain scope to reduce the possibility of ambiguity.

### 3.2 A Model of Natural Language Processing (NLP)

A model of a natural language processing system is shown in Figure 3.1.

In Figure 3.1, the “Parser” module is the basic component of an NLP system. It reads an input sentence and performs, step by step, the lexical, syntactic and semantic analysis. The output of a parser is a formal logical representation of the input sentence. The module “Reasoning System” then adds the pragmatic information to it. Finally, the “Action System” module converts the derived logical form to a suitable command language for a computer execution or to generate a natural language response to the end user.

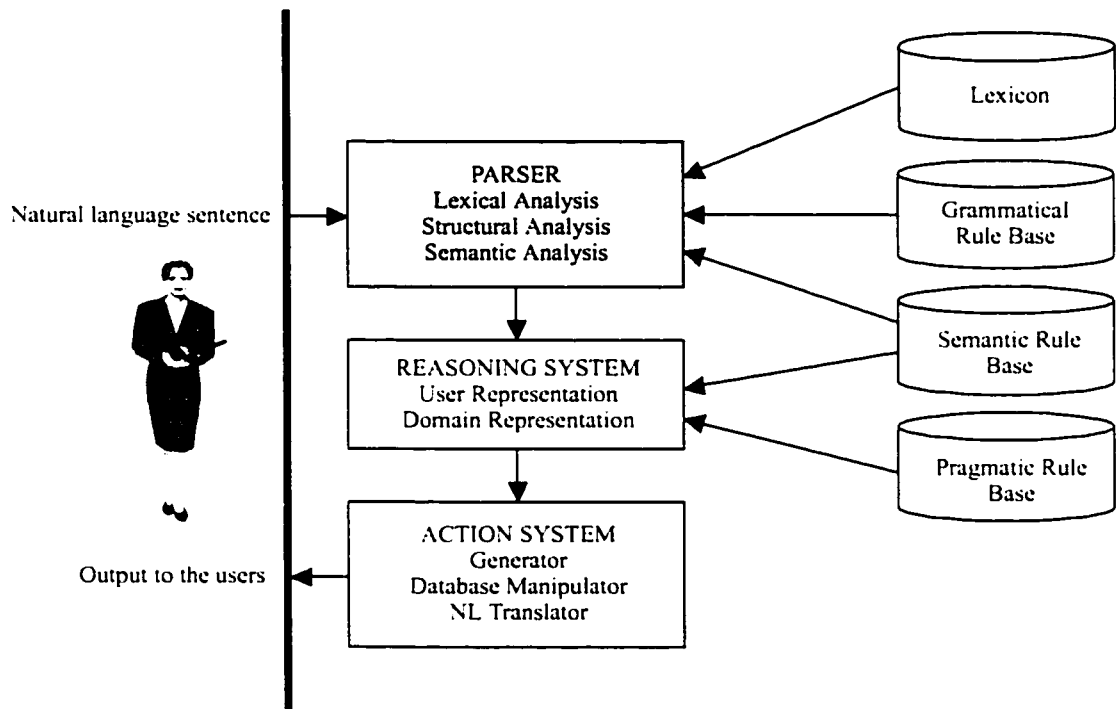


Figure 3.1: A Model of an Natural Language Processing System

### 3.2.1 Lexical Analysis

In lexical analysis, the NLP system divides the input sentence into words or tokens, and looks up in the lexicon to see if it recognizes the word or token. The lexicon is a collection of the words and their related information such as the word's root, forms, lexical categories, semantic meanings, its prefixes, suffixes, and the rules describing how these "pieces of information" combine etc. A simple lexicon is shown in Table 3.1. It is a database containing the word itself, its type, the verb's root and the word's form.

Word	Word Type	Verb Root	Word Form
a	Determiner		
house	Noun		
street	Noun		
jump	Noun		
jump	Verb	jump	
jumping	Verb	jump	ing
is	Auxiliary		
to	Preposition		
where	WH word		

Table 3.1: An Example of a Lexicon

In Table 3.1, word "jump" has two types: noun and verb. This is a kind of lexical ambiguity. It may cause two or more parse trees to be derived in the syntactic analysis.

### 3.2.2 Syntactic Analysis

In syntactic analysis, the parser analyzes the sentence to determine its structure according to a pre-defined grammar. The output of a parser is one or more "parse trees", each of which represents a syntactic structure of the input sentence. If more than one parse tree is derived, it means that the input sentence contains ambiguities or multiple interpretations. Semantic analysis is the next step to solve these ambiguities according to the analysis of the semantic meanings.

### 3.2.2.1 Context-Free Grammar

The context-free grammar (CFG) was first introduced by Chomsky [8] to describe the **phrase structure** of natural language sentences. The formal definition of the context-free grammar is:

Assume

- (1)  $V$  is a finite set of non-terminal symbols.
- (2)  $T$  is a finite set of terminal symbols, where  $N \cap T = \emptyset$ .
- (3)  $P$  is a finite subset of  $V \times (V \cup T)^*$ .
- (4)  $S$  is a symbol of  $V$  designated as the start symbol.

A grammar  $G = (V, T, S, P)$  is said to be context-free if all productions in  $P$  have the form

$$A \rightarrow x$$

where  $A \in V$  and  $x \in (V \cup T)^*$ . A language  $L$  is said to be context-free if and only if there is a context-free grammar  $G$  such that  $L = L(G)$ .

The CFG is important because its formalism is powerful enough to be able to describe most of the structures in natural languages and yet it is restricted enough so that efficient parsers can be built to analyze sentences.

An example of context-free grammar is shown in Figure 3.2.

The grammar can be mapped to the parse tree shown in Figure 3.3.

### 3.2.2.2 Recursive Transition Networks(RTN)

The recursive transition network represents the context-free grammar in a graphical form. An example of it is shown in Figure 3.4. Within the network, the nodes represent “states” and the “arcs” between nodes are labeled. The labels are assigned grammatical

- (1)  $S \rightarrow NP VP$
- (2)  $VP \rightarrow \text{verb NP}$
- (3)  $NP \rightarrow \text{name}$
- (4)  $NP \rightarrow \text{deter noun}$
- (5)  $\text{noun} \rightarrow \text{"man"}$
- (6)  $\text{name} \rightarrow \text{"Gloria"}$
- (7)  $\text{deter} \rightarrow \text{"the"}$
- (8)  $\text{verb} \rightarrow \text{"saw"}$

Figure 3.2: An Example of Context-Free Grammar

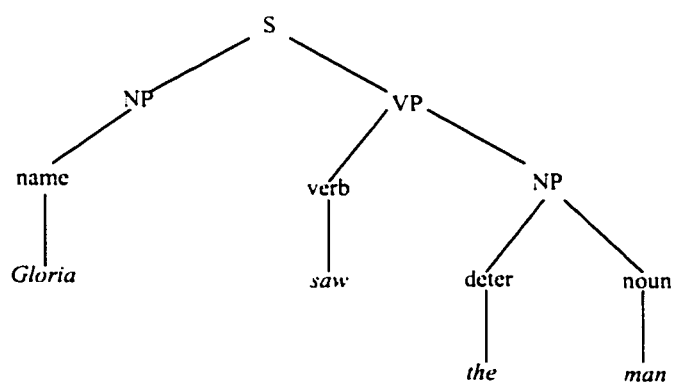


Figure 3.3: Parse Tree of the input "*Gloria saw the man.*"

tags such as noun, verb, NP and VP etc. For the non-terminal items such as NP and VP, other networks are defined to represent their CFGs; thus, the network is recursive.

### 3.2.2.3 Top-Down Parsing and Bottom-Up Parsing Techniques:

Parsing techniques for natural languages can be summarized into two categories, one is **top-down** and the other is **bottom-up**. In top-down parsing, the parser reads a CFG rule from the left-hand side. In the process of top-down parsing, the left-hand side symbol (goal) is rewritten into right hand symbols (sub-goals) sequentially until a sequence of terminal symbols is obtained. Top-down parsing usually combines a technique called backtracking in order to recursively derive the parse tree. A top-down parsing algorithm with backtracking technique can also be used in recursive transition networks [47], [48]. In Figure 3.4's RTN, the parser reads the net from the start state  $S_0$  and browses the net until it reaches the end state  $S_7$ . If it encounters an arc labeled with a non terminal symbol, it will jump into that non-terminal item's sub-network, travel in it, and then return back to the original network until the sub-network's end state is reached. The parsing is a success if it reaches the original network's end state. A parse tree is derived at the same time as the output of the parsing process.

Top-down parsing techniques have the advantage that they will never consider word categories in positions where they could not occur in a legal sentence. However, when an error occurs in the early stage of the top-down parsing, the parsing process will be very time consuming.

Since bottom-up parsing methods start by looking at the input and then building up the larger structures, those structures that cannot lead to the input words will never be built. However, it costs time to consider all senses of each word and construct useless structures that could never lead to a legal sentence.

In contrast to top-down parsers, bottom-up parsers read the CFG rules in a reverse direction. The top-down parsing algorithm uses a data structure called **chart** as a book-



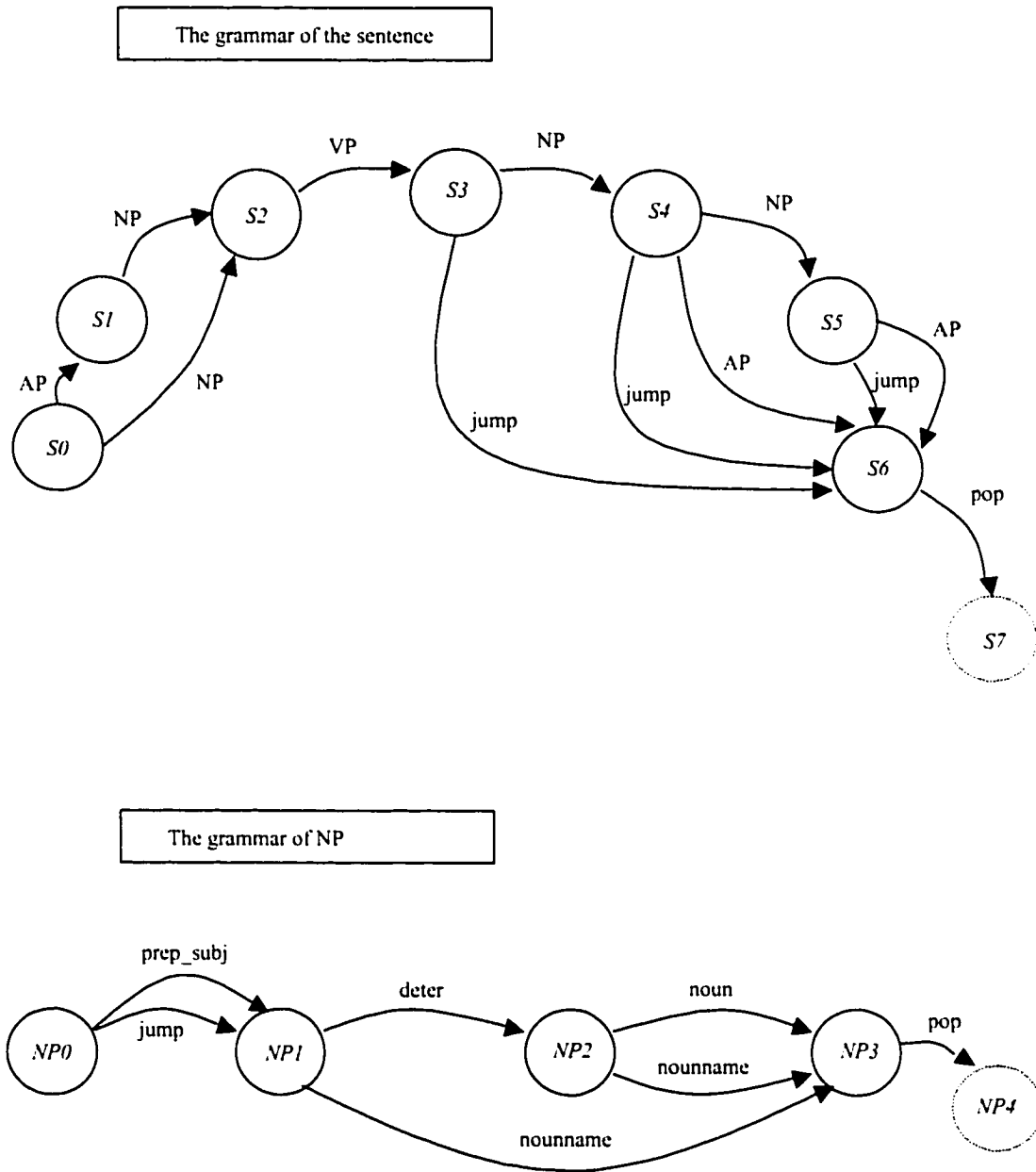


Figure 3.4: An Example of Recursive Transition Network

keeping storage for all the information produced while parsing. This information includes the positions of the words, the new structures derived from the sentence and the rules that have partially matched but are not complete yet (active arcs). Unlike top-down parser, the bottom-up parser looks for the CFG rules from the sentence start symbol, the chart parser tries to match every possible CFG rules to the word it reads. It then records the current status into the chart and continues to read the next word until it finds a matched rule.

#### 3.2.2.4 LR Parsing

LR parsing method is a bottom-up parsing technique. It was first introduced to handle a small subset of context-free grammar sufficient for programming languages. This method is considered the most efficient parsing technique right now. The LR parser consists of two parts, a driver routine and a parsing table (shown in Figure 3.5 [2]). The parsing table is the place to store the grammars. When the parser works, it reads the input stream from left to right with one symbol at a time. Being guided by the parsing table, it constructs a rightmost derivation.

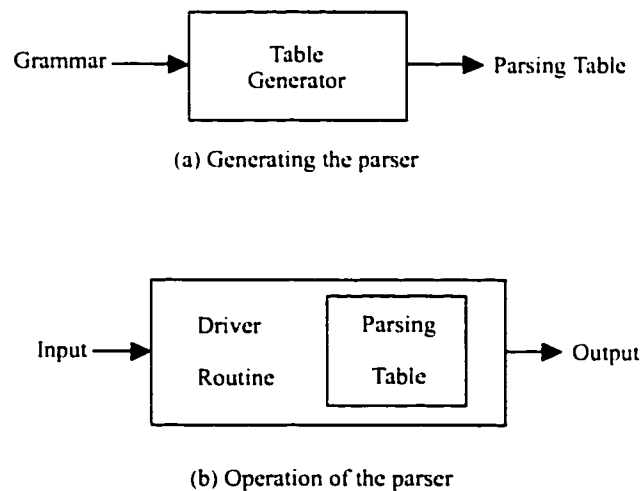


Figure 3.5: Generating an LR Parser

There are many different parsing tables that can be used in a LR parser for a given grammar. Simple LR(SLR) is the easiest one to implement. Unfortunately, it may fail to

produce a table for certain grammars on which the other methods would succeed. Canonical LR(CLR) is the most efficient one and will work on a very large class of grammars. LALR(lookahead LR) uses a more compact LR table than the one used in CLR and its efficiency lies in between SLR and CLR. The generation of the parsing table is a too complex work to be completed by hand. Fortunately, many parser generators have been developed by people to generate a LR parser. YACC [22] and CUP [21] are such kind of parser generators.

LR parser is an attractive method to parse natural languages because of the following two reasons:

- Its efficiency makes the response time of the interactive natural language interface short enough to keep the dialogues between the system and the end users “smooth”.
- The LR grammar, although is only a subset of context-free grammars, is still close to the natural language grammars. Thus, it is possible to extend the LR parser a little bit to make it suitable to parse natural languages. In fact, many efforts have been done to extend LR parsing methods to make them more suitable for natural language parsing. Tomita’s GLR parsing algorithm [43] is one of the examples.

Alternatively, if we restrict the natural language to “a small scope” in both lexicon and syntax, we can also use the LR parser to parse that restricted natural language.

### **3.2.3 Semantic Analysis**

The target of semantic analysis is to add each word’s meaning into the parse trees derived from the syntactic analysis, resolve the possible ambiguities that may exist in the natural language input, and combine these meanings to form a logical form. The logical form is the formal representation of the input sentence that is suitable for execution by a computer. It uses mathematical methods to make the sentence’s semantics clear and unambiguous. Logic, production rules and semantic networks are some typical means to represent this logical form. Case grammar, which is introduced by Fillmore [15], is used to establish a bridge between the sentence syntactic structure and its logical form.

### 3.2.3.1 Case Grammar

Case grammar was first proposed by Fillmore in 1968 [15] and was widely accepted by AI researchers [7]. The idea behind case grammar is that every sentence has an underlying representation of its meaning. This representation includes the verb and the various noun phrases related to the verb. It actually describes the **case relationships** between the verb and its nouns in the sentence (shown in Table 3.2).

Agent(A)	the instigator of the event
Counter-Agent(C)	the force or resistance against which the action is carried out
Object(O)	the entity that moves or changes or whose position or existence is in consideration
Result(R)	the entity that comes into existence as a result of the action
Instrument(I)	the stimulus or immediate physical cause of an event
Source(S)	the place from which something moves
Goal(G)	the place to which something moves
Experience(E)	the entity which receives or accepts or experiences or undergoes the effect of an action

Table 3.2: Fillmore's Case System

Consider the following two sentences

S1: "*John opened the door with the key.*"

S2: "*The door was opened by John with the key.*"

Using the case grammar, we designate "*John*" as the agent, "*door*" as the object, and "*key*" as the instrument for the verb "*opened*". Suppose the verb "*open*" can be used in the following four ways: (A - agent, O - object, I - instrument)

- The door opened. (O)
- John opened the door. (A, O)
- The wind opened the door. (I, O)
- John opened the door with the key. (A, O, I)

We can observe that when “*open*” is used, its object must appear in the surface sentence whereas the agent and instrument are optional. If we find the relationships between the sentence surface structure and the deep case grammar, we can translate the surface sentence into its case structured form and then easily interpret it into the logical form.

### 3.3 Habitability

The notion of habitability was introduced by Watt in 1968 [44] in order to measure how productively a formal language can be used to express everything the user needs without exceeding the language boundary. A language is considered habitable if the user can express everything that is required for a given task using only sentences allowed by the language. In order to satisfy the habitability criterion, the designers of the natural language interface should study the task(s) at hand, make the interface as easy to use as possible and keep users stay within the conceptual, functional, syntactical and lexical limitations of the habitability [38].

The concept of habitability is important in the natural language interface domain, because with the current technology, any natural language interface can only be restricted into a bounded problem domain, thus restricts its habitability into a limited scope. Therefore, the following two problems become challenging and unavoidable in the development of a natural language interface.

Problem1: How to bring users from somewhere outside the language boundary back to lie within the habitability region of the language (cure)?

Problem2: How to avoid users to use the language conceptually, functionally, syntactically or lexically exceeding the habitability limits (prevention)?

## Chapter 4

### A Restricted Natural Language for Policy Capturing

In this chapter, we propose a user-centered design approach for policy capturing. In the design of user interfaces to software systems, the “user centered approach” is gaining importance [14], [40]. The key goal of this approach is to produce usable systems that are highly “friendly” to the end users. **Usability**, the most important concept of Human Computer Interaction(HCI), is concerned with the following key factors [18]:

- (1) effectiveness: the accuracy and completeness with which specified users can achieve specified goals in particular environments.
- (2) efficiency: the resources expended in relation to the accuracy and completeness of goal achieved.
- (3) satisfaction: the comfort and acceptability of the work system to its users and other people affected by its use.

To support this definition, our design objective for the interfaces for policy capturing focuses on the following items:

- (1) ease of use
- (2) ease of learning how to perform the intended tasks using the new software
- (3) adapting the interface to match different classes of user needs
- (4) naturalness in the software-human interactions.

To achieve the targets of ease of use and ease of learning, we propose a Restricted Natural Language (RNL) Interface for the end users to dynamically define and change the policies in a policy-based system.

#### 4.1 User-Centered Design for Policy Capturing

Consider this scenario: A person is using a newspaper reading software to browse several local and global newspapers. The software is installed on a PC with a connection to Internet. It retrieves the newspapers' contents from various databases by Internet and reads them to the end user by means of text-to-speech algorithms. Before using this software, the user first sets up some policies and preferences about what contents he or she wants to retrieve. For example, he or she wants to read the international news in Globe and Mail after the terrorist attacks happened in the U.S in September 11th, 2001 and the sports news in Montreal Gazette when the 2002 Olympics begins at Salt Lake City. Then, he or she specifies the following policies for the system to maintain.

- (1) Read every international news from Globe and Mail after September 11th, 2001.
- (2) Read every sports news from Montreal Gazette after the 2002 Olympics begins.

We find that the policies the end user sets up are simple policies. These policies may change dynamically depending on the end users' intention. In addition, these policies are personalized and may be different according to the different users' interests. The users who set up these policies are, most likely, the casual users and may not be familiar with the formal languages which are used to set up the policies in most of the policy-based systems. In such a situation, using any kind of Policy Description Languages or logic-based policy representations are not appropriate for users who have low tolerance to learn artificial languages to perform their tasks (set up the policies); instead, using a Restricted Natural Language (RNL) to set up the policies becomes a good choice; and if the system uses a speech recognition technique to allow user input via speech, the usability of the software can be further expanded.

The RNL interface provides an easy-to-use and easy-to-learn environment for the end users to input and modify the dynamically changing policies. It is especially useful under the following conditions:

- The policies are changed dynamically and frequently.
- The policies are simple. They are basically describing the action between one subject and one object. We will explain in the following chapters why the policies should be kept simple.
- The policies are suitable to be described in natural language. In some situations, using natural language to express a policy could be much more complex and longer than using some artificial languages. Let's see an example policy defined in a Message Notification System (MNS) [23].

A MNS is a system that can deliver messages to the recipients. When inputs arrive to MNS, the system generates a notification to the recipient and is sent through one of the delivery media like pager, or voice mail. The recipient would call back to MNS to acknowledge that the notification was received. A 1-800 number is provided by the system for receiving acknowledgements. If the intended recipient does not acknowledge the receipt of a message, the MNS server would send another notification to the next receiver on a list who is supposed to receive that message as an alternate receiver.

A sample policy in this case is expressed using a script language (a kind of artificial languages):

When Contact arrives:

If Contact.priority = "critical":

Then Notify ( message = Contact.message; priority = 1; severity = 1:

recipient1 = "Contact.recipient"; delivery1 = pager; delay1=10;

recipient2 = "Wang"; delivery2 = pager; delay2=20;



recipient3 = "David"; delivery3 = pager; delay3=30;

The policy says when a Contact arrives the system will check its priority. If the priority is "critical", the system will first deliver the notification and the Contact.message to the first recipient listed above. If the recipient doesn't acknowledge the notification in the listed delay time, the system will sequentially send the notification and the Contact.message to the next recipient listed above until the system receives an acknowledgement.

The above explanation could be seen as a natural language version of the sample policy. We observe that using natural language, this policy has to be expressed in a paragraph instead of a simple sentence. In addition, the list of the recipients is not clearly stated in the textual description.

In a complex situation like the above, the use of natural language is not straightforward.

The tasks that our intended users do are simple. Defining, modifying and reviewing are three basic tasks that users need to do with each policy. We believe a Restricted Natural Language (RNL) could be used for such simple tasks.

**T1: define a policy.**

Step 1: Find out in which step should this policy be executed.

Step 2: Input the policy by keyboard typing or speaking. Wait the system to check the policy. If it passes the check, the system saves the policy. If not, go to Step 3.

Step 3: Correct the errors by re-entering part of the policy, until no error remains. The system then saves the policy.

**T2: modify a policy.**

Step 1: Find out which policy needs to be modified.

Step 2: Re-input the policy by keyboard typing or speaking. Wait for the system to check the policy for consistency. If it passes the check, the system saves the policy and exits.

Step 3: Correct the errors by re-entering part of the policy, until no error remains. The system then saves the policy.

### **T3: review a policy.**

Find out which policy the user wants to review, retrieve it from the storage and display it.

Despite supporting the above tasks, a RNL interface should also have the ability to transfer the restricted natural language to a formal language without losing any information. Since a natural language (even when it is restricted) is too hard to be understood by machines, a formal language must be used as the communication tool inside the systems. As for representing policies, we have known that PDLs and logic-based approaches can form such a tool. In this thesis, we choose a logic-based formal language as the system's internal language to represent policies. As a result, a translator that translates the RNL statement to the formal logic-based representation becomes a necessary part of the RNL interface.

Based on the above analysis we say that if we construct a RNL as the communication language between the end user and the system, such a RNL interface would be easy to use and to learn. In addition, such an interface matches the different levels of users' needs and retains a natural way in the software human interactions.

To keep things simple, we restrict each policy to be sentence-based. That is to say, we use no more than one sentence to express one policy. Moreover, we observe that policies are executed in a pre-determined sequence, that is to say, some policies should be executed earlier than other policies. (Explained in Section 6.3.1.) We use an ordinal number that

tags the policies to represent this sequence. In this way, we avoid the need to express a complex time constraint in each policy. Also, it saves us from giving an undetermined physical time constraint in an abstract level policy. As we pointed out in Chapter 2, we leave this problem to the further policy refinement steps.

## 4.2 Level I - The Restricted Natural Language

In our two levels of policy specifications, the Restricted Natural Language is in the higher level, which is used for human-computer interactions.

In this case, the restricted natural language contains a limited vocabulary and a simple syntax. It is used for expressing simple policies. The characters of the “simple” policies can be described as follows:

- One policy can be expressed in one “simple” sentence.
- As defined in Section 2.1, the policies described in this thesis are a set of actions, which are triggered under certain conditions, acting on the target objects. That is to say, the statement like “Root is a member of the super user group” is not a valid policy in our scope because assertions like this do not explicitly result in any action. Alternatively, the statement “The super user group is combined with root and system administrator” can be seen as a valid policy, because the action “combine” is explicitly specified.
- Each policy contains only one subject, one action and one object.

The restricted natural language is restricted in three dimensions.

Restriction-1: The vocabulary or the lexicon is limited and it is specific to the problem domain. The vocabulary doesn’t record any word or the meaning of the word outside the problem domain. In addition, different verb forms may not be recorded into the vocabulary if they are not useful for expressing the policies. For example, for the verb “attend”, we only record its root form “attend” and

its singular form “attends”. Other forms such as the past tense “attended” are ignored since we never use a past tense to express a policy in the chosen domain of applications. At the same time, we recognize “attend” and “attends” as synonyms because we don’t need to remember the semantic meaning behind the verb tenses and the singular/plural forms. We also define some **open words** and some **closed words**. A set of **open words** can be expanded by the end user at any time; that is new words can be added or existing words in the set can be deleted. A set of **closed words** contains the words fixed at the time when the vocabulary is created. We make the closed words into a small set. In this way we restrict the lexicon of the RNL.

Restriction-2: the syntactic structures used to form sentences are kept simple and few in number. There are several reasons for choosing a simple syntax for our RNL.

- The target we are expressing is the first consideration. The policies we express are simple. Every policy contains only one subject, one action and one object. There’s no need to use complex sentence structures.
- Sentence length is another factor we consider. Research in [39] indicates that sentence ambiguity increases exponentially with respect to sentence length. Usually, sentences whose lengths are more than twenty words can be considered stylistically poor, and can lead to serious problems in comprehension [31]. For that reason, we use simple sentence structures as well as limit the sentence length.
- The use of simple syntax also gives us a chance to use efficient parsers to reduce the response time of the system. Since real natural language sentences may have many interpretations and the real natural language parser has to handle all of them, it is not surprising that a real natural language grammar is usually big and the real language parser is not as

efficient as compiler parsers such as a LALR parser. Because we restrict the sentence syntax to a simple set, we can use an efficient LALR parser to parse our RNL sentences. Although ambiguities will still exist, they are quite limited. We can solve them by repeatedly generating different parse trees and filtering the wrong ones out in the semantic analysis stage, and at the same time, we can assure acceptable response time.

**Restriction-3:** The semantics of the statements are expressed in terms of an underlying object-oriented model of the domain. Any object outside the scope of the domain is not included in this model. Obviously the RNL interface cannot understand the semantics that depend on such excluded objects. In another aspect, the restriction of the object-oriented model itself imposes a restriction of the RNL's semantics. An object-oriented model is suitable to express the semantics like "an action happens/an action does not happen". However, it is not appropriate to express the semantics such as "to what degree or how an action happens" or "until what degree an action may cause such a result" etc. Consequently, our RNL is restricted in expressing such kind of semantics.

In this way, we draw a "clear boundary" for our RNL. Once a natural language obtains clear lexical, syntactic and semantic scopes, ambiguity problems will be greatly decreased and a practical RNL processor is relatively easier to develop.

### **4.3 Level II - A Formal Logic-based Language (FR)**

As the lower level in our two levels of policy specifications, a Formal Logic-based Language is used to store the policies inside the policy-based system and for further automatic processes. This formal language is an essential part of the policy-based systems for internal representation. To translate the RNL into the formal language becomes one of our RNL interface's functional requirements.

Compared to the RNL, the formal language is unambiguous. All the automatic

processing such as policy conflict analysis, policy verification, policy refinement etc. is based on this language. That's why a formal policy representation plays a fundamental role in the policy-based system architecture.

In this thesis, we choose a temporal logic-based approach as the formal language to represent the policies. Our approach is combined with two parts: the object-oriented model to represent the entities and the actions in the policies, and the logic to express the obligation/authorization policies, the constraints and the relationships between two actions. The details of how to construct such a FR is explained in Chapter 5.

#### **4.4 A Methodology to Construct a Restricted Natural Language for Policy Capturing**

We use the following seven steps to construct a Restricted Natural Language for the problem domain.

Step1: Define the problem domain by capturing "adequate" number of use case scenarios from one or more background materials and creating a sample set of policy statements. The number of use case scenarios can be seen "adequate" when all the concepts inside the problem domain are covered by these use case scenarios. The use case maps or UCMs can be drawn at this step in order to visually representing the causal use case scenarios combined with structures.

Step2: Create an object-oriented model for the problem domain. In this stage, the objects are clustered into classes; the attributes and the methods of classes are defined; the relationships and interactions between classes are determined.

Step3: Based on the cases selected in the problem domain (Step1) and the domain expertise, construct a vocabulary.

Step4: Select a minimal set of simple-to-use syntactic structures for the intended RNL.

Step5: Verify by manual methods to see if all the use case scenarios can be expressed in RNL, that is to verify if the RNL satisfies the “habitability” requirement. (The concept of habitability has been explained in Section 3.3.)

Step6: Develop a formal logical specification based on the semantics created by the object-oriented model.

Step7: Develop a mapping from the RNL to the formal logic-based representation (FR) to suit the further refinement or enforcement steps.

To explain in detail how the above step-by-step procedure works, we give two examples in Chapter 5.

#### 4.5 The Expandability of the RNL

The RNL is designed as an expandable language. That is, the end users are allowed to add words into the vocabulary or expand the object-oriented model to accommodate more objects. The RNL can be expanded in two ways.

**Expand the Vocabulary:** End users can add new words into the vocabulary at any time they want. However, they can only modify or add to the set of open words. Since closed words are often related to the structures and semantics of the sentences, modifying them may result in re-programming of the whole RNL analyzer. So, closed words are fixed at the time when the vocabulary is created and do not allow to be modified.

**Expand the Object-Oriented Model:** Users can add objects, attributes or operations into the existing object-oriented model. As a result, they have to expand the vocabulary to fit the modified model.

**The Syntax is not allowed to expand:** We believe that by expressing the policies in an appropriate way, most of the policies can be described using the existing syntax.

Furthermore, any “small” modification of the syntax will cause a re-generation of the RNL parser. For the above reasons, we do not allow users to modify the syntax.

Expanding a RNL may cause a serious ambiguity problem. Let’s see an example to explain how expanding a RNL can cause an ambiguity problem. Suppose we already have a RNL. Inside the vocabulary, we have a word “committee” representing the object “the graduate studies committee”. Now we add a new object “the examining committee” into the object-oriented model and use the same word “committee” to represent this object in the vocabulary. An ambiguity arises. Does the word “committee” mean “the graduate studies committee” or “the examining committee”? In this thesis, we try to use noun phrases to avoid this kind of ambiguity problems. In this example, we delete the word “committee” from the vocabulary and add two noun phrases to represent the two different objects: “graduate studies committee” and “examining committee”. In Chapter 6, we will discuss more about this.



## Chapter 5

### Case Study

In this chapter, we use two examples to explain how to construct a RNL using the methodology described in Chapter 4. In the first example, we explain how to construct a RNL for a Virtual Thesis Office (VTO) step by step. Then, we briefly introduce our second example, a Virtual Call Center (VCC).

#### 5.1 Virtual Thesis Office (VTO)

A Virtual Thesis Office (VTO) is a simplified system that handles procedures related to the submissions and examinations of students' theses like the real thesis office at Concordia University does. The thesis office accepts students' theses, appoints examining committees, schedules the defence dates, holds defences, and makes the final decisions for the theses according to a set of pre-defined policies. The policies of the VTO have the same contents as the regulations set up for the real thesis office.

In this chapter, we will set up a policy management system for the VTO. In order to accept users' statement of policies in natural language, we will construct a RNL for the VTO system using the methodology proposed in Chapter 4.

##### 5.1.1 Step1: Define the Problem Domain

*Define the problem domain by capturing "adequate" number of use case scenarios and creating a sample set of policy statements. The use case maps or UCMs can*

*be drawn at this step in order to visually representing the causal use case scenarios combined with structures.*

#### 5.1.1.1 Use Case Maps

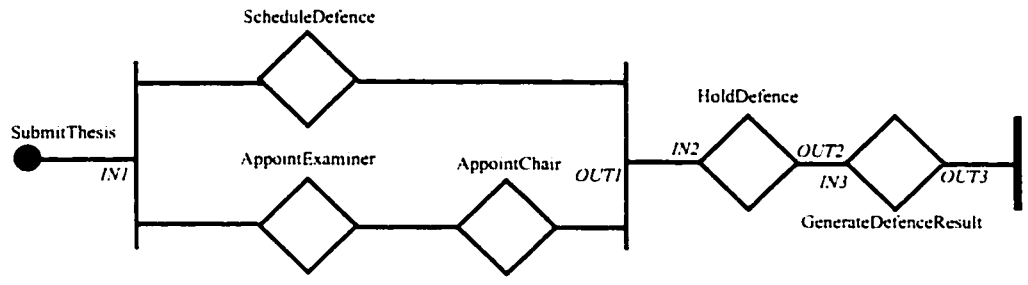
To construct a RNL for the VTO system, the first thing we need to do is to define the problem domain. As mentioned above, our problem domain will be restricted to the aspects of a VTO system's regulations for dealing with masters' theses. We first find out the document about the examinations of the master's theses from the Concordia University's Graduate Calendar (1999-2000) and develop as many use case scenarios as we can. Two use case maps are drawn in Figure 5.1 and Figure 5.2. The notations used in these UCMs are as per [Ref] or Ref.

To schedule a master's thesis examination, the candidate needs to first submit the thesis to the VTO (start point *SubmitThesis* in Figure 5.1(a) Root Map). The VTO then needs to schedule the defence date (stub *ScheduleDefence* in Figure 5.1(a) Root Map). At the same time, the Graduate Committee needs to appoint an examining committee for the thesis examination (stub *AppointExaminer AppointChair* in Figure 5.1(a) Root Map). After that, the VTO needs to hold the defence somewhere in the school with the attendance of the student himself/herself and the examiners (stub *HoldDefence* in Figure 5.1(a) Root Map). Finally, the examiners need to give a defence result to the student (stub *GenerateDefenceResult* in Figure 5.1(a) Root Map).

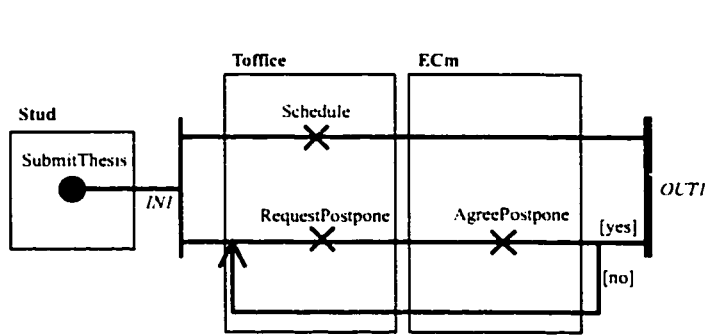
The plug-ins in Figure 5.1(b), (c), (d), (e), (f) explain in detail how the VTO system works.

(b) [Plug-in *ScheduleDefence*]: It describes how the VTO system decides a thesis defence date. (Figure 5.1(b))

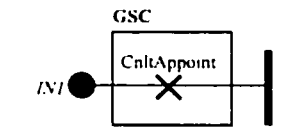
*Stud*: (Figure 5.1(b)) Object *Stud* represents the candidate or student who submits the thesis.



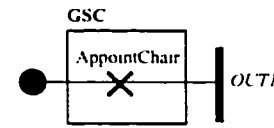
(a) Root Map



(b) ScheduleDefence Plug-in

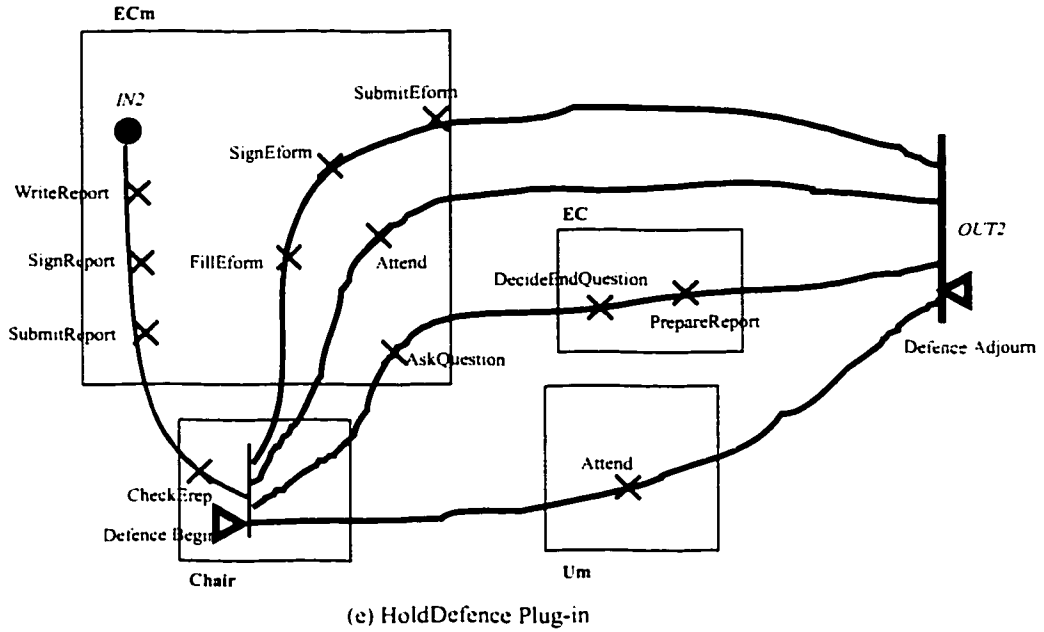


(c) AppointExaminer Plug-in



(d) AppointChair Plug-in

Figure 5.1: The Use Case Map for the VTO system



- [c1]: accepted with major modification
- [c2]: no decision made
- [c3]: accept as submitted, accepted with minor modifications or rejected

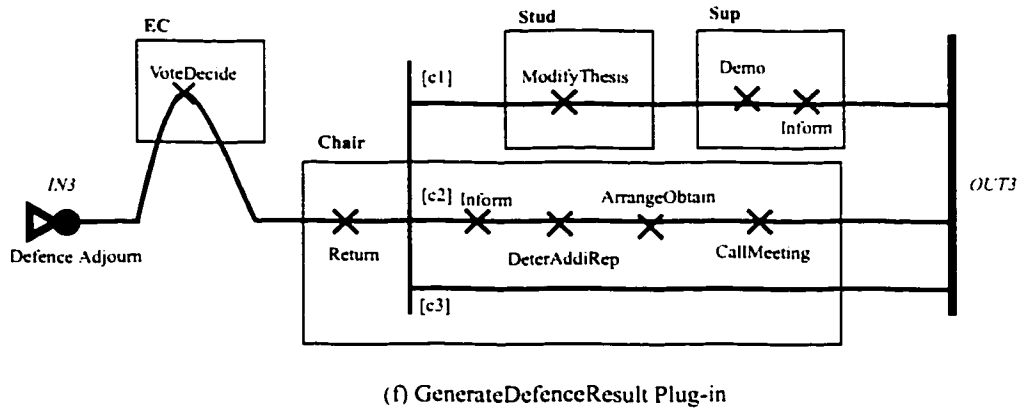


Figure 5.2: The Use Case Map for the VTO system - Continue

*SubmitThesis*: (Figure 5.1(b)) At the start point, the object *Stud* submits his/her thesis to the VTO.

*IN1*: (Figure 5.1(b)) At point *IN1*, the VTO system receives the thesis submitted from the candidate.

*Toffice*: (Figure 5.1(b)) This is an object name representing the thesis office.

*Schedule*: (Figure 5.1(b)) Object *Toffice* performs this action to schedule the thesis defence date.

*ECm*: (Figure 5.1(b)) This is an object name representing the member of the examining committee.

*RequestPostpone*: (Figure 5.1(b)) Object *Toffice* performs action *RequestPostpone* to require a postponement to the current thesis defence date.

*AgreePostpone*: (Figure 5.1(b)) When *Toffice* requests a postponement, object *ECm* should perform action *AgreePostpone* to reply *Toffice* if they agree upon the postponement. If the answer is "yes", then the new thesis defence date is determined. If the answer is "no", then *Toffice* should propose a new request to the postponement.

*OUT1*: (Figure 5.1(b)) At this end point, the defence date is determined.

(c) [**Plug-in *AppointExaminer***]: It shows what actions are performed to appoint the examiners. (See Figure 5.1(c).)

*IN1*: (Figure 5.1(c)) At point *IN1*, the VTO system receives the thesis submitted from the candidate.

*GSC*: (Figure 5.1(c)) This is an object name representing the graduate studies committee.

*CnltAppoint*: (Figure 5.1(c)) *GSC* performs the action *CnltAppoint* to appoint examiners in order to organizing an examining committee for the thesis defence.

(d) [**Plug-in *AppointChair***]: It shows what actions are performed to appoint a chair for the examining committee. (See Figure 5.1(d).)

*GSC*: (Figure 5.1(d)) This is an object name representing the graduate studies committee.

*AppointChair*: (Figure 5.1(d)) *GSC* performs the action *AppointChair* to appoint a chair for the examining committee.

*OUT1*: (Figure 5.1(d)) At the end point *OUT1*, Both the examiners and the chair have been appointed.

(e) [**Plug-in *HoldDefence***]: It describes how the thesis defence is held. (See Figure 5.2(e).)

*ECm*: (Figure 5.2(e)) This is an object name representing the member of the examining committee.

*IN2*: (Figure 5.2(e)) This is the start point of plug-in *HoldDefence*.

*WriteReport*: (Figure 5.2(e)) Object *ECm* performs the action *WriteReport* for preparing a report for the thesis before the defence begin.

*SignReport*: (Figure 5.2(e)) Object *ECm* should sign the report he/she prepared by performing the action *SignReport*.

*SubmitReport*: (Figure 5.2(e)) After signing the report, object *ECm* should submit the report to the chair by performing the action *SubmitReport*.

*Chair*: (Figure 5.2(e)) This is an object name representing the chair of the examining committee.

*CheckErep*: (Figure 5.2(e)) Before the thesis defence begin, the chair should check if all the examiners submit the reports by performing the action *CheckErep*.

**Time Stamp - *DefenceBegin***: At this point the defence begins. (See the triangle in Figure 5.2(e).)

*FillEform*: (Figure 5.2(e)) During the defence, the object *ECm* should fill an evaluation form by performing the action *FillEform*.

*SignEform*: (Figure 5.2(e)) Object *ECm* should sign the evaluation form after filling it out. The action for sign this evaluation form is *SignEform*.

*SubmitEform*: (Figure 5.2(e)) After the evaluation form is signed, *ECm* should submit it to the chair by performing the action *SubmitEform*.

*Attend in ECm*: (Figure 5.2(e)) Object *ECm* should attend the thesis defence by performing the action *Attend*.

*AskQuestion*: (Figure 5.2(e)) During the defence, the *ECm* will ask questions by doing the action *AskQuestion*.

*EC*: (Figure 5.2(e)) *EC* is an object name representing the examining committee.

*DecideEndQuestion*: (Figure 5.2(e)) During the defence, *EC* will make the decision that no further question is needed by performing the action *DecideEndQuestion*.

*PrepareReport*: (Figure 5.2(e)) A report should be prepared by *EC* for the thesis defence.

*Um*: (Figure 5.2(e)) *Um* is an object name representing the member of the university.

*Attend in Um*: (Figure 5.2(e)) any *Um* can attend the thesis defence by performing the action *Attend*.

*OUT2*: (Figure 5.2(e)) At the end point of *HoldDefence* plug-in, the defence is adjourned.

**Time Stamp - DefenceAdjourn**: This is the time stamp representing the time the defence being adjourned. (See the triangle in Figure 5.2(e).)

**(f) [Plug-in GenerateDefenceResult]**: It describes how the thesis defence result is generated. (See Figure 5.2(f).)

*IN3*: (Figure 5.2(f)) At the start point plug-in *GenerateDefenceResult*, the defence is adjourned.

**Time Stamp - *DefenceAdjourn***: This is the time stamp representing the time the defence being adjourned. (See the triangle in Figure 5.2(f).)

*EC*: (Figure 5.2(f)) This is an object name representing the examining committee.

*VoteDecide*: (Figure 5.2(f)) Object *EC* should vote to make the decision for the thesis defence by performing the action *VoteDecide*.

*Chair*: (Figure 5.2(f)) This is an object name representing the chair of the examining committee.

*Return*: (Figure 5.2(f)) The *Chair* should return all the reports and evaluation forms to the thesis officer by performing the action *Return*.

*Inform in Chair*: (Figure 5.2(f)) If there's no decision made, the *Chair* should inform the candidate that the decision is pending by performing the action *Inform*.

*DeterAddiRep*: (Figure 5.2(f)) If there's no decision made yet, object *Chair* should determine what additional information is required for the examining committee by performing the action *DeterAddiRep*.

*ArrangeObtain*: (Figure 5.2(f)) After doing the action *DeterAddiRep*, the *Chair* should then arrange to obtain this additional information by performing the action *ArrangeObtain*.

*CallMeeting*: (Figure 5.2(f)) When additional information is obtained, the *Chair* should call another meeting to reach a decision for the thesis defence by performing the action *CallMeeting*.

*Stud*: (Figure 5.2(f)) This is an object name representing the candidate.

*ModifyThesis*: (Figure 5.2(f)) If the thesis is accepted with major modification, the object *Stud* should modify the thesis by performing the action *ModifyThesis*.



*Sup:* (Figure 5.2(f)) This is an object name representing the candidate's supervisor.

*Demo:* (Figure 5.2(f)) After *Stud* modified his/her thesis (*ModifyThesis*), the *Sup* should demonstrate the modification to the examining committee by performing the action *Demo*.

*Inform in Sup:* (Figure 5.2(f)) *Sup* should also inform the thesis officer that the modification is approved by performing the action *Inform*.

*OUT3:* (Figure 5.2(f)) At this end point, the decision of the defence is made. If the defence is accepted with major modification, the required modification has been done. If there's no decision reached during the defence adjourn period, the late decision has been made at this point. This is the end point of the all thesis examination processes.

#### 5.1.1.2 Sample Policies

A set of sample policies is given below. These are derived manually from the real procedures followed by the Concordia Thesis Office.

P1 Upon receipt of the thesis from the Thesis Office, the graduate studies committee of the program in which the candidate is enrolled appoints an examining committee in consultation with the thesis supervisor. The examining committee consists of a minimum of three, and a maximum of five members. The candidate's supervisor is a member of the examining committee.

P2 In the programs where there is a thesis supervisory committee, any or all members may be named as members of the examining committee, subject to the policy of the program concerned.

P3 SIP students must have one external examiner on their committee.

- P4 Students in the M.A.Sc. programs in Engineering must have one examiner from outside their department.
- P5 The defence normally shall be scheduled within a period no fewer than two, and no more than five, weeks from the submission of the thesis.
- P6 The parties concerned may agree upon a postponement.
- P7 The examining committee, and the thesis defence examination, will be chaired by a person appointed by the program's graduate studies committee. The chair will act as a neutral person.
- P8 Each member of the examining committee must submit a written report on the thesis to the chair before the defence.
- P9 Any member of the University is free to attend a master's oral thesis defence.
- P10 The chair will give priority to questions from members of the examining committee.
- P11 Only members of the faculty of the University may be recognized by the chair.
- P12 The chair adjourns the examination when the examining committee decides that further questioning is unnecessary.
- P13 The deliberations of the examining committee are held in camera, that is to say, only appointed members of the examining committee are present.
- P14 It is the responsibility of the chair to see that a report on the examination is prepared before the committee adjourns.
- P15 The examiner's evaluation forms and the examination report must be signed and returned to the thesis officer.

P16 Four decisions are open to the examining committee, voting to be based on a simple majority. The thesis can be accepted as submitted, accepted with minor modifications, accepted with major modifications, or rejected. Minor modifications are defined as corrections which can be made immediately to the satisfaction of the supervisor. Major modifications are defined as corrections requiring further research, or structural changes, or other substantive revision.

P17 When a thesis is accepted with major modifications, a precise description of the modifications must be included in the examining committee's report along with a date for their completion. It is then the responsibility of the candidate's supervisor to demonstrate to the examining committee that the required modifications have been made. The candidate's supervisor should inform the Thesis Officer in writing that the modifications have been approved.

P18 If the examining committee is not prepared to reach a decision concerning the thesis at the time of the thesis defence, it is the responsibility of the chair to determine what additional information is required by the committee to reach a decision, to arrange to obtain this information for the committee, and to call another meeting of the committee as soon as the required information is available. It is also the chair's responsibility to inform the candidate that the decision is pending.

We notice that the use case maps do not provide detailed information for the operational aspects and the sample policy statements stay in an abstract level. This is because both the use case maps and the sample policies are derived from the real world policies used for humans. Policies stayed in the abstract level are already clear enough for humans to execute. The operational details of these policies, in contrast, are redundant and only expected to be useful for the machines. Since our target is to construct a RNL for expressing the policies, either the abstract level or operational level description is adequate. The policy

refinement step should take the responsibility for mapping the policies from the abstract level to the operational level.

### 5.1.2 Step2: Create the Object-Oriented Model

*Create an object-oriented model for the problem domain. In this stage, the objects are clustered into classes; the attributes and the methods of classes are defined; the relationships and interactions between classes are determined.*

Once the domain is clearly bounded, we will create an object-oriented model from the use case scenarios and the sample policies set up in Step1.

First we abstract objects from the sample policies by analyzing the nouns. As shown in Figure 5.3, for example, class *ECm* is an abstraction of “member of the examining committee”, class *Um* is an abstraction of “member of the university”, and class *Extm* is the abstraction of “external examiner”. Some abstract classes are added in order to make the generalization relationships complete (shown in Figure 5.3). For example, *Doc* and *Person* are abstract classes added by the designer. A complete explanation of the class names is listed following the Figure 5.3:

#### LIST OF CLASSES:

**Class Person** is added by the designer. It represents all the persons in the VTO model.

**Class Extm** is an abstraction of “external examiner” in sample policy *P3*.

**Class ECm** is an abstraction of the “member of the examining committee” in sample policy *P2*.

**Class Sup** is an abstraction of the “thesis supervisor” in sample policy *P1*.

**Class Chair** is an abstraction of the “Chair” in sample policy *P1*.

**Class SupCm** is an abstraction of the “member of the supervisory committee” in sample policy *P2*.

**Class Um** is an abstraction of the “member of the university” in sample policy *P9*.

**Class Outdeptm** is an abstraction of the “examiner from outside the department” in sample policy *P4*.

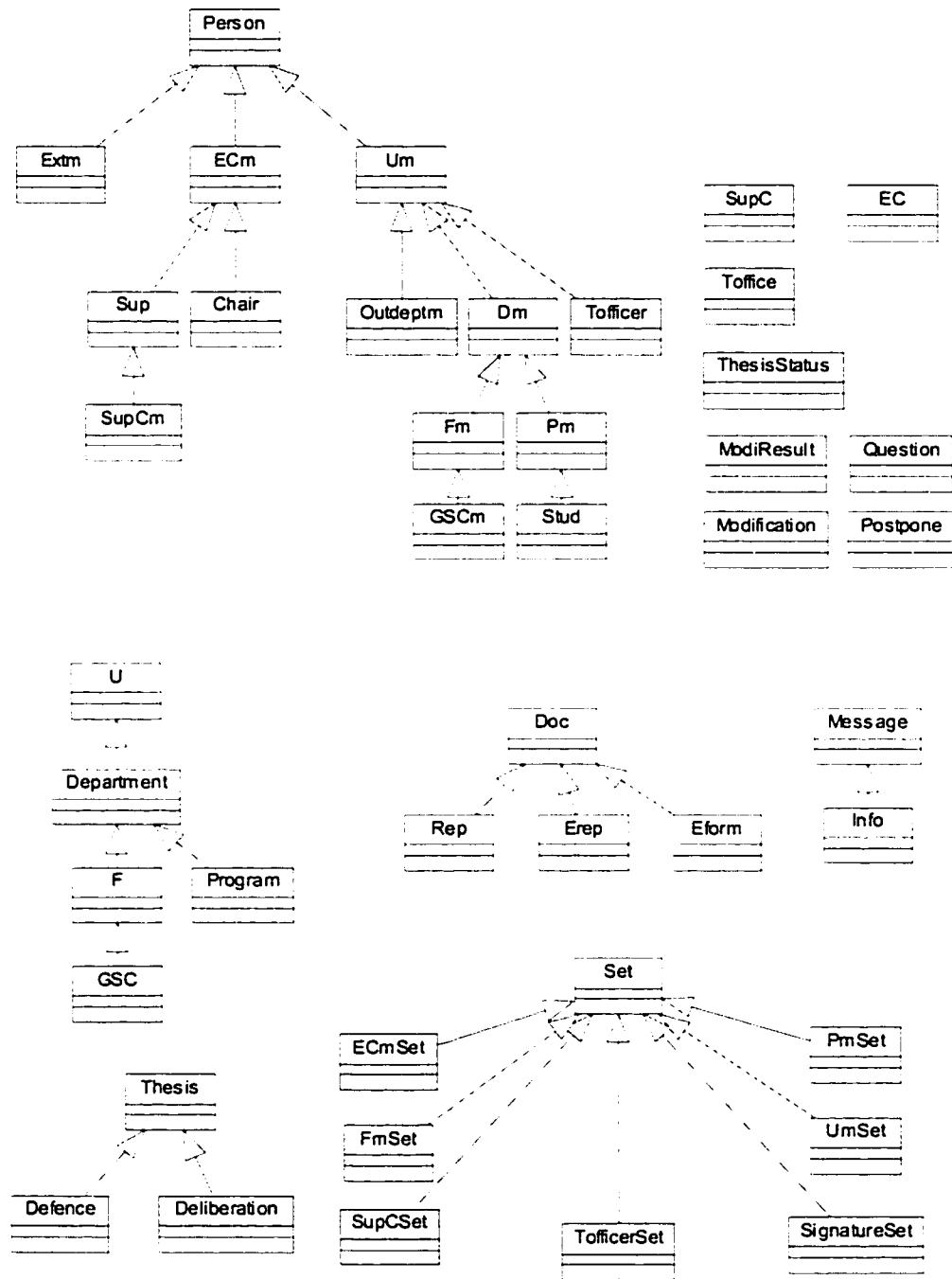


Figure 5.3: Class Diagram for the VTO Model and its generalization relationships

**Class Dm** is an abstraction of the “member of the department” added by the designer.

**Class Fm** is an abstraction of the “member of the faculty” added by the designer.

**Class GSCm** is an abstraction of the “member of the graduate studies committee” added by the designer.

**Class Pm** is an abstraction of the “member of the program” added by the designer.

**Class Stud** is an abstraction of the “candidate” or “student” in sample policy *P1*, *P3* or *P4*.

**Class Tofficer** is an abstraction of the “thesis officer” in sample policy *P15*.

**Class SupC** is an abstraction of the “supervisory committee” in sample policy *P2*.

**Class EC** is an abstraction of the “examining committee” in sample policy *P1*.

**Class Toffice** is an abstraction of the “thesis office” in sample policy *P1*.

**Class ThesisStatus** is an abstraction of “the four decisions made by the examining committee regarding the thesis defence” in sample policy *P16*.

**Class ModiResult** means the result of the modification. It is abstracted from the sentence group “the required modifications have been made” in sample policy *P17*.

**Class Modification** represents the modification the student made for his/her thesis according to the examining committee’s requirement. It is abstracted from the sentence group “the required modifications have been made” in sample policy *P17*.

**Class Question** represents the questions asked in the thesis defence. It is abstracted from the sentence “the chair will give priority to questions from members of the examining committee” in sample policy *P10*.

**Class Postpone** is an abstraction of the “postponement” in sample policy *P6*.

**Class U** is an abstraction of the “university” in sample policy *P9*.

**Class Department** is an abstraction of the “department” in sample policy *P4*.

**Class F** is an abstraction of the “faculty” in sample policy *P11*.

**Class GSC** is an abstraction of the “graduate studies committee” in sample policy *P1*.

**Class Program** is an abstraction of the “program” in sample policy *P1*.

**Class Doc** is a generalization of all the documents used in the VTO model. It is added by the designer.

**Class Rep** is an abstraction of the “written report” in sample policy *P8*.

**Class Erep** is an abstraction of the “report on the examination” in sample policy *P14*.

**Class Eform** is an abstraction of the “evaluation form” in sample policy *P15*.

**Class Message** is a generalization of all the messages send/receive between the chair, examination committee, supervisor, student, and the thesis office. It is added by the designer.

**Class Info** is an abstraction of the “additional information” in sample policy *P18*.

**Class Thesis** is an abstraction of the “thesis” in sample policy *P1*.

**Class Defence** is an abstraction of the “thesis defence” in sample policy *P9*.

**Class Deliberation** is an abstraction of the “deliberation” in sample policy *P13*.

**Class Set** is an abstract class added by the designer. It represents a group of objects.

**Class ECmSet** is an abstract class added by the designer. It represents the group of the examining committee member.

**Class FmSet** is an abstract class added by the designer. It represents the group of the faculty members.

**Class SupCSet** is an abstract class added by the designer. It represents the group of the supervisory committee members.

**Class TofficerSet** is an abstract class added by the designer. It represents the group of the thesis officers.

**Class SignatureSet** is an abstract class added by the designer. It represents the group of the signatures.

**Class UmSet** is an abstract class added by the designer. It represents the group of the university members.

**Class PmSet** is an abstract class added by the designer. It represents the group of the program members.

We then fill out the attributes of the classes and summarize the operations by analyzing the use cases and the sample policies. The relationships between the classes are added at the same time. A sample class *ECm* is shown in Figure 5.4. A complete class list is shown in Appendix B.

#### **LIST OF ATTRIBUTES:**

**Ecm\_rep:** *Ecm\_rep* is an attribute of class *ECm*. It represents the report that each member of the examining committee must fill before attending the defence.

**signature:** *signature* is an attribute of class *ECm*. It represents the signature of each examiner.

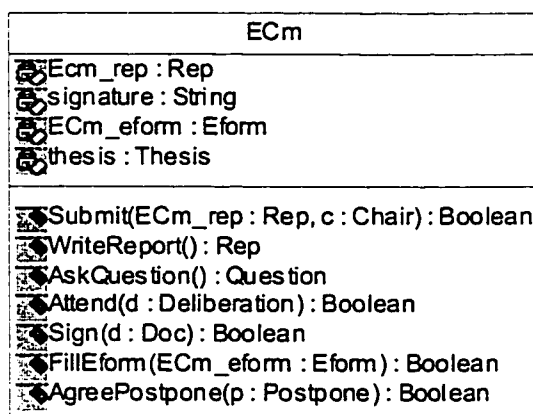


Figure 5.4: ECm Class

**ECm\_iform:** *ECm\_iform* is an attribute of class *ECm*. It represents the “evaluation form” that each examiner should submit after the defence.

**thesis:** *thesis* is an attribute of class *ECm*. It represents the thesis the candidate defends.

**Submit:** *Submit* is a method of class *ECm*. It represents the action described in P8 “Each member of the examining committee must submit a written report on the thesis to the chair before the defence”.

**WriteReport:** *WriteReport* is a method of class *ECm*. It represents the action the examiner *write the report*.

**AskQuestion:** *AskQuestion* is a method of class *ECm*. It represents the action that people *ask questions* in the defence.

**Attend:** *Attend* is a method of class *ECm*. It represents the action that people *attend* the defence.

**Sign:** *Sign* is a method of class *ECm*. It represents the action that the examiners *sign* the documents like the evaluation forms etc.

**FillEform:** *FillEform* is a method of class *ECm*. It represents the action that the examiners *fill the evaluation forms*.

**AgreePostpone:** *AgreePostpone* is a method of class *ECm*. The semantics is coming from P6 “The parties concerned may agree upon a postponement”. It represents the action that the examiners state their opinion whether they *agree upon the defence postponement*.

### 5.1.3 Step3: Construct a Vocabulary

*Based on the cases selected in the problem domain and the domain expertise, construct a vocabulary.*



Once the object-oriented model is created, we can begin to construct the RNL's vocabulary. Three types of information should be determined at this stage. They are **name**, **type** and **meaning**.

**Name:** The name records the information about how the words look like. In our vocabulary, both a single word and a word group can be names. A word group is a group of words syntactically acts as a single word in a sentence and has a semantic meaning. We call a word group as a **segment** in our RNL.

**Type:** The type information is used for the syntactical analysis. We introduce 8 word types in our RNL. They are:

- (1) **deter** (determiner),
- (2) **noun/nounname** (a segment acts as a noun),
- (3) **aux** (auxiliary),
- (4) **verb/verbname** (a segment acts as a verb),
- (5) **prep** (preposition),
- (6) **prep\_subj** (a special kind of preposition that can only be placed before the indirect object. Here is an example. "Each examiner must return the evaluation form *to* the chair." "*to*" is a prep\_subj.),
- (7) **conj** (conjunction) and
- (8) **adv** (adverb).

**Meaning:** The meaning records the semantic information for each word. In our RNL, most of the semantics or meanings come from the object-oriented model. For example, the name "*member of the examining committee*" means the object *ECm* in the object-oriented model. So the class name *ECm* is recorded as the meaning of "*member of the examining committee*". When the object-oriented model is expanded, this part of vocabulary should also be expanded. The words or segments in this part

of vocabulary are called “open words”. (See Section 4.2 for the definition.) On the other hand, the semantics or meanings of certain words are not coming from the object-oriented model. These names have fixed unique meanings in practice (“closed words”. See Section 4.2 for the definition). Table 5.1 is a sample list of such words.

Segment Type	Word	Be Translated To
Determiner	<i>“the, this, only, each”</i>	$\forall$
	<i>“an, a, any, any or all”</i>	$\exists$
Auxiliary	<i>“may, be free to”</i>	$\diamond$
	<i>“must, can, should, will, shall”</i>	$\square$
Preposition	<i>“before”</i>	{constraint} $t <$
	<i>“after”</i>	{constraint} $t >$
	<i>“for, in, upon”</i>	{trigger}
Conjunction	<i>“when, as soon as, if”</i>	{trigger}
Adverb	<i>“also”</i>	<i>“empty”</i> (meaningless in RNL)

Table 5.1: A List of Words with Fixed Unique Meanings

We use at least one word or segment to represent each class and do the same thing to attributes and operations of classes. In this way, a RNL’s vocabulary is constructed. Figure 5.5 shows a part of such a vocabulary. It lists the verbs or verb segments for the VTO’s RNL. For the purpose of simplicity, the RNL parser doesn’t provide mechanisms to recognize different forms of verbs. The different forms of the verbs are seen as the synonyms of the original verbs. For example, word “attends”, “attended” and “attending” are translated to be the synonyms of word “attend”.

#### 5.1.4 Step4: Construct the Syntax

*Select a minimal set of simple-to-use syntactic structures for the intended RNL.*

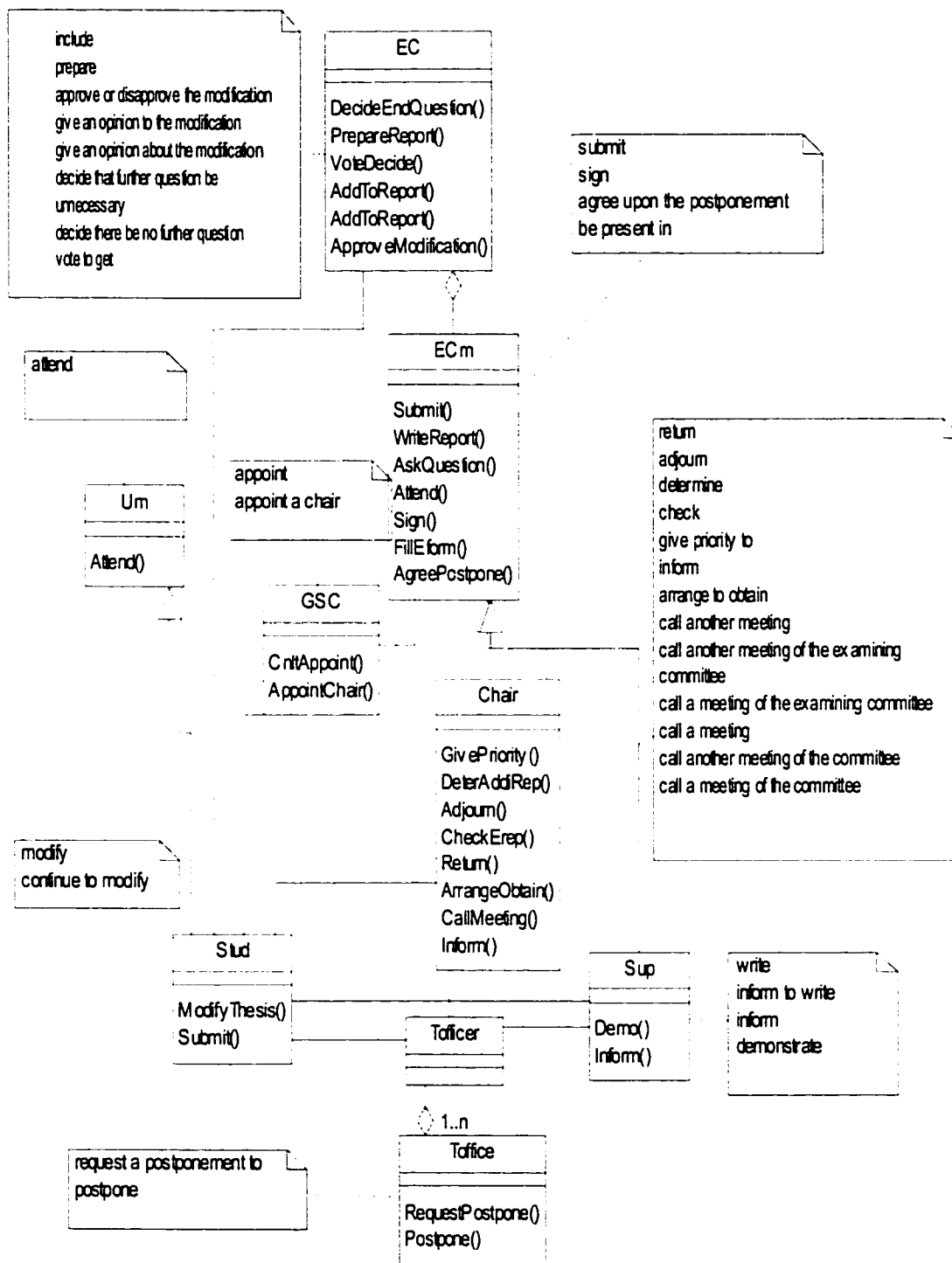


Figure 5.5: The Verbs Listed in the Vocabulary

#### 5.1.4.1 The Grammar for the sentences

Compared to the flexible syntax of natural English sentences, the sentence structures of policies are quite limited and fixed because basically a policy is a well organized complete sentence describing what action should be done under certain conditions. In this thesis, we selected three groups of sentence structures and a total of eight grammar rules as our RNL's syntax by analyzing the sample policy statements. We believe that using these eight grammar rules one can describe most of the policies (conditions and actions) in a natural way. Figure 5.6 shows these eight grammar rules. Figure 5.7 presents some examples for these eight grammar rules.

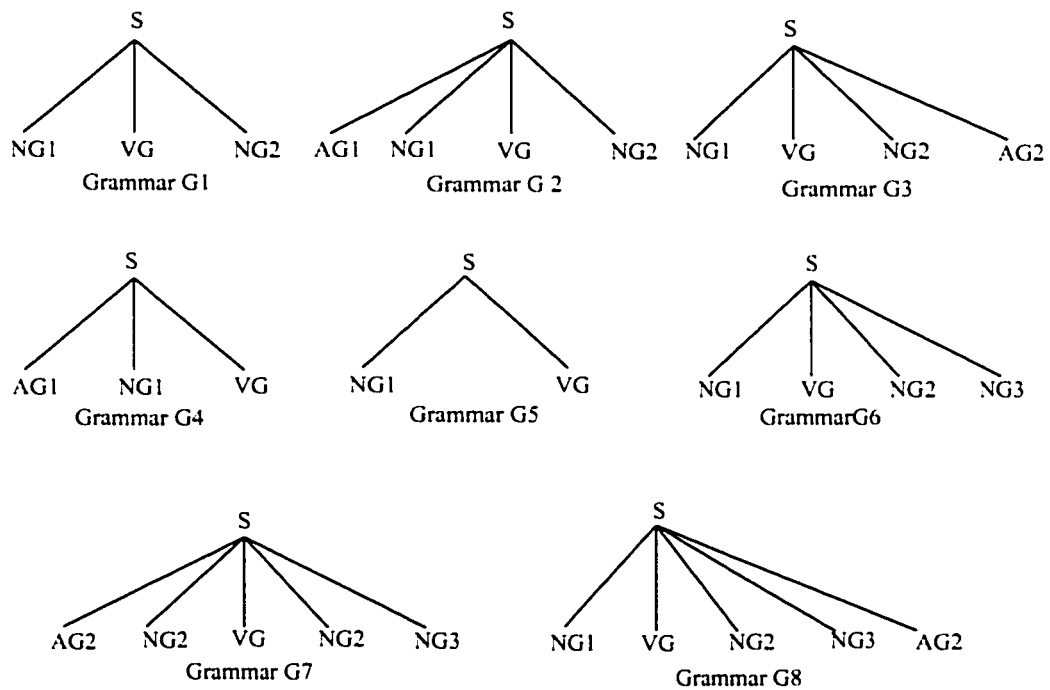
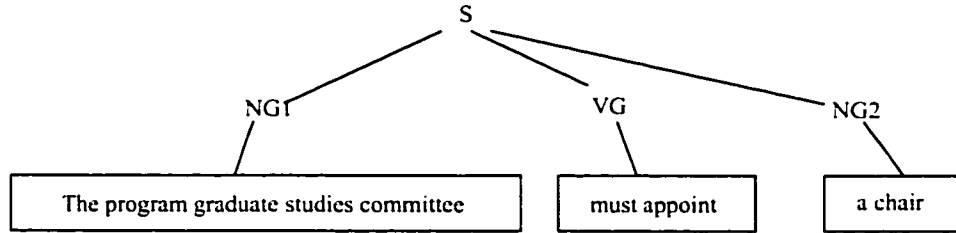


Figure 5.6: Eight Grammar Rules for VTO System's RNL

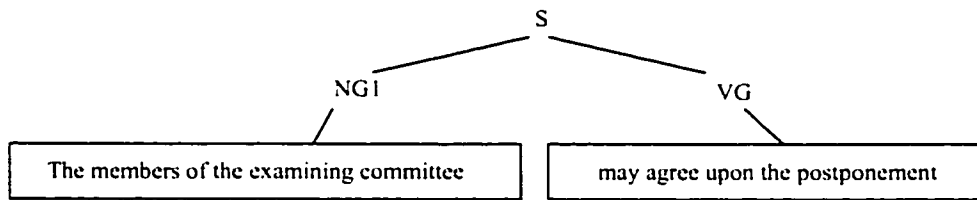
The following notations are used in Figure 5.6.

**S:** The sentence.

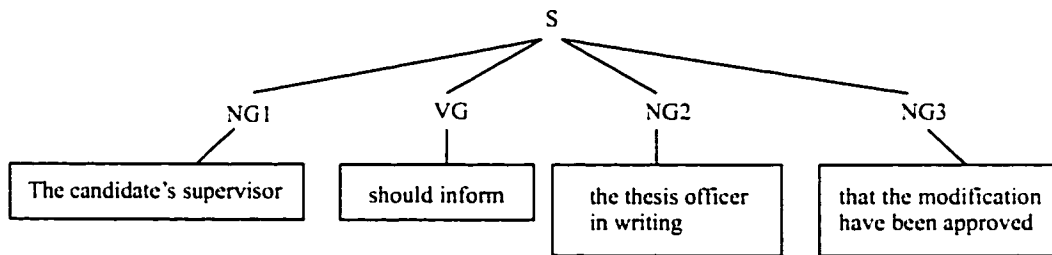
**NG:** Noun group. It is a group of words which acts as a noun. It could be the subject or object in the sentence, and it is an elementary part of a sentence.



An Example of Grammar G1



An Example of Grammar G 5



An Example of Grammar G6

Figure 5.7: Some Examples for VTO System's RNL Grammar

**VG:** Verb group. It is a group of words which acts as a verb. It acts as the predicate of a sentence and is an elementary part of a sentence.

**AG:** Adjective/Adverb group. It is a group of words which acts as an adjective or adverb. It could be the modifier of the noun group, the adverbial modifier or the complement of the sentence. It is not an elementary part of a sentence.

#### 5.1.4.2 The Grammar for NG, VG and AG

Sentences are made up of groups and groups consist of segments or groups. The following BNF formulas describe the syntax we use to construct NG, VG and AG.

```

NG ::= PREP.SUBJ DETER NOUN
      | PREP.SUBJ DETER NOUNNAME
      | PREP.SUBJ NOUNNAME
      | DETER NOUN
      | DETER NOUNNAME
      | NOUNNAME:

```

```

VG ::= VERB
      | VERBNAME
      | AUX VERB
      | AUX VERBNAME
      | AUX ADV VERB
      | AUX ADV VERBNAME:

```

```

AG ::= CONJ NG VG
      | CONJ NOUNNAME
      | PREP NOUNNAME:

```

### 5.1.5 Step5: Manually Verify the RNL's Habitability

*Verify by manual methods to see if all the use case scenarios can be expressed in RNL, that is to verify if the RNL satisfies the "habitability" requirement.*

To test if the vocabulary is sufficiently large to express every aspect of the policies inside the VTO problem domain and if the syntax set is appropriately selected to construct the policy sentences, we manually rewrote the sample policies of the VTO system using the vocabulary we constructed and the grammar rules selected. Following is our conclusion:

- Most of the information contained in the use case scenarios is incorporated into the policy statements. However, because of the limitation of the RNL syntax and vocabulary, policies have to be rewritten into simple sentences. As a result, the total number of policies becomes almost double. In Appendix C, a complete set of the rewritten policies is provided. <sup>1</sup>
- The RNL does not cover all the complexities shown in the use case scenarios. When things become too complex, it is the responsibility of the end users to rewrite the difficult things into simple sentences that can be expressed in the RNL. Examples of this can be found in Appendix C. In Appendix C, the end user rewrote 18 sample policies into 34 simpler policy sentences in order to express them in RNL (P1 to P18 in Section 5.1.1.2). If there are no difficult things to express in RNL, the RNL is 100% habitable.

---

<sup>1</sup> Two cases are not incorporated into the policy statements because the syntax and the vocabulary of RNL are too simple to cover them. Following is a list of such exceptions.

- \* In sample policy *P1* "...The examining committee consists of a minimum of three, and a maximum of five members. ...", the cardinality information "a minimum of three, and a maximum of five" cannot be expressed in RNL because RNL doesn't have mechanisms to analyze such a complex determiner.
- \* In sample policy *P5* "the thesis officer should normally schedule the defence within a period no fewer than two, and no more than five, weeks from the submission of the thesis.", RNL is not complex enough to express AG "within a period no fewer than two, and no more than five, weeks from the submission of the thesis".

### 5.1.6 Step6: Develop a Formal Logic-based Policy Specification

*Develop a formal logical specification based on the semantics created by the object-oriented model.*

Our Formal Representation (FR) consists of two parts. One is the object-oriented model. It is used to represent the entities and the actions in the policies. The other is based on logic, which is the glue to stick the entities and the actions together. The latter also adds the obligation/authorization information, the constraints and the trigger conditions to the policies.

#### 5.1.6.1 Objects

A set of objects having common attributes can be grouped into a class. We describe the attributes, operations, association relationships and generalization relationships in the object-oriented model which is explained in Section 5.1.2. For example, the class *ECm* shown in Figure 5.4 describes the common attributes of all the members of the examining committee. *ECm.signature* is an attribute of class *ECm*. *ECm.Submit* is an operation of class *ECm*.

In policy definition, an instance of the class is used as the object or subject of the policy sentence. If we define *ecm* as an instance of class *ECm* in policy “P8: *Each member of the examining committee must submit a written report on the thesis to the chair before the defence.*”, then *ecm* represents “(a particular) member of the examining committee”. *ecm.signature* is an attribute of *ecm*. *ecm.Submit* is an operation of *ecm*.<sup>2</sup>

#### 5.1.6.2 Modality $Do(p, action)$

We define a modality  $Do(p, action)$  to represent the following structure “Somebody takes some action”.  $p$  means a person or an agent who does the action.  $action$  is the

<sup>2</sup> In this thesis a class name starts with a capital letter (*ECm*) and the instance of that class starts with the same letter but in lower case (*ecm*).



operation the person or a software agent takes. Actions come from the “operations” of the object-oriented model. “ $Do(p, action)$ ” can be TRUE or FALSE indicating whether the agent has performed the action or not. Here is an example of the use of this structure. In order to represent information that “the examiner signs the evaluation form” we can use the following form:

$$Do(ecm, Sign(ecm.Ecm\_eform))$$

### 5.1.6.3 Temporal Operators $\Box_t(f)$ and $\Diamond_t(f)$

To represent the obligation/authorization information and the time constraints involved in the policies, we use temporal operators  $\Box_t(f)$  and  $\Diamond_t(f)$ .

Assume  $f$  as a formula such as a modality  $Do(p, action)$ , we define  $\Box_t(f)$  as “ $f$  is necessary at time  $t$ ”;  $\Diamond_t(f)$  denotes “ $f$  is possible at time  $t$ ”.

For example, to represent the obligation information in the policy “the examiner must sign the evaluation form before the defence”, we can use the following expression.

$$\Box_{t < thesis.defence.defence\_begin\_time} Do(ecm, Sign(eform))$$

If the above policy is changed to “the examiner may sign the evaluation form before the defence”, we then change our expression as follows:

$$\Diamond_{t < thesis.defence.defence\_begin\_time} Do(ecm, Sign(eform))$$

### 5.1.6.4 The Formal Policy Representation (FR)

We also use the quantifiers  $\forall$ ,  $\exists$  and the “if...then” operator “ $\rightarrow$ ” in our formal representation. The complete expression of the policy “The examiner must sign the evaluation form before the defence” now becomes

$$\forall ecm, \forall eform, \square_{t < thesis.defence.defence\_begin\_time} Do(ecm, Sign(eform))$$

Another example shows how the “if...then” operator “ $\rightarrow$ ” works.

**Example:** “As soon as the additional information is available, the chair shall call another meeting of the committee.”

$$\forall ec, ec.info \neq NULL \rightarrow$$

$$\forall ec.chair, \square Do(ec.chair, CallMeeting())$$

#### 5.1.6.5 The List of FR Format Policies in VTO

In corresponding to the 34 rewritten RNL policies, the 34 FR format policies is generated and listed here.

P1.1: Upon receipt of the thesis from the thesis office, the graduate committee of the program in which the candidate is enrolled should appoint the candidate’s supervisor to be a member of the examining committee.

$$\exists tofficer, \exists stud Do(stud, Submit(tofficer, stud.thesis)) \rightarrow$$

$$\forall stud.gsc, \forall stud.sup \square Do(stud.gsc, CnltAppoint(stud.sup))$$

P1.2: Upon receipt of the thesis from the thesis office, the graduate committee of the program in which the candidate is enrolled should appoint a person to be a member of the examining committee in consultation with the thesis supervisor.

$$\exists tofficer, \exists stud Do(stud, Submit(tofficer, stud.thesis)) \rightarrow$$

$$\forall stud.gsc, \exists stud.sup, \exists person \square Do(stud.gsc, CnltAppoint(stud.sup, person))$$

P2: In the program where there is a thesis supervisory committee, the graduate committee of the program may appoint any or all members of the thesis supervisory committee to be the member of the examining committee.

$$\forall stud, stud.supc.count > 2 \rightarrow$$

$$\forall stud.gsc, \exists person, \exists supcm \diamond Do(stud.gsc, CnltAppoint(supcm, person))$$

P3: The graduate studies committee must appoint an external examiner for SIP students.

$$\forall stud, stud.program.program\_name = "SIP" \rightarrow$$

$$\forall stud.gsc, \exists stud.sup, \exists extm \square Do(stud.gsc, CnltAppoint(stud.sup, extm))$$

P4.1: The graduate studies committee must appoint an examiner outside the department if the student is in the M.A.Sc program in Engineering.

$$\forall stud, stud.program = "M.A.Sc" \rightarrow$$

$$\forall stud.gsc, \exists outdeptm \square Do(stud.gsc, CnltAppoint(outdeptm))$$

P4.2: The graduate studies committee must appoint an examiner outside the department for the student in the M.A.Sc program in Engineering.

$$\forall stud, stud.program = "M.A.Sc" \rightarrow$$

$$\forall stud.gsc, \exists stud.sup, \exists outdeptm \square Do(stud.gsc, CnltAppoint(stud.sup, outdeptm))$$

P6.1: The thesis office may request a postponement addressed to the members of the examining committee.

$$\forall toffice, \exists thesis, \forall ecm \diamond Do(toffice, RequestPostpone(ecm, thesis))$$

P6.2: The members of the examining committee may agree upon the postponement.

$$\forall ecm, \exists postpone \diamond Do(ecm, AgreePostpone(postpone))$$

P6.3: If the members of the examining committee agree upon the postponement, the thesis office should postpone the thesis defence.

$$\forall ecm, \exists postpone Do(ecm, AgreePostpone(postpone)) \rightarrow$$

$$\forall toffice, \exists postpone, \forall ecm.thesis.defence$$

$$\square Do(toffice, Postpone(ecm.thesis.defence, postpone))$$

P7: The program graduate studies committee must appoint a chair.

$$\forall gsc, \exists sup, \exists chair \Box Do(gsc, CnlAppoint(sup, chair))$$

P8: Each member of the examining committee must submit a written report on the thesis to the chair before the defence begins.

$$\forall ecm, \exists ecm.ECm\_rep, \forall chair \\ \Box_{t < thesis.defence.defence\_begin\_time} Do(ecm, Submit(ecm.ECm\_rep, chair))$$

P9: Any member of the University is free to attend a master's oral thesis defence.

$$\exists um, \exists thesis.defence \Diamond Do(um, Attend(thesis.defence))$$

P10: The chair will give priority to questions from members of the examining committee.

$$\forall chair, \forall question \Box Do(chair, GivePriority(question))$$

P12: The chair adjourns the examination when the examining committee decides that further questioning is unnecessary.

$$\forall ec \Box Do(ec, DecideEndQuestion()) \rightarrow \\ \forall ec.chair, \forall ec.thesis, \Box Do(ec.chair, Adjourn(ec.thesis.defence))$$

P13: Only appointed members of the examining committee are present in the deliberations of the examining committee.

$$\forall ecm, \forall thesis.deliberation Do(ecm, Attend(thesis.deliberation))$$

P14.1: The examining committee should prepare a report before the committee adjourns.

$$\forall ec, \exists erep \Box_{t < thesis.deliberation.adjourn\_end\_time} Do(ec, PrepareReport(erep))$$

P14.2: The chair should check the examining committee's report.

$$\forall chair, \forall erep \Box Do(chair, CheckErep(erep))$$

P15.1: Each examiner must sign the examiner's evaluation form.

$$\forall ecm, \forall eform \Box Do(ecm, Sign(eform))$$

P15.2: Each examiner must sign the examination report.

$$\forall ecm, \forall erep \square Do(ecm, Sign(erep))$$

P15.3: The chair must return the examiner's evaluation forms to the thesis officer.

$$\forall chair, \forall eform, \forall tofficer \square Do(chair, Return(eform, tofficer))$$

P15.4: The chair must return the examination report to the thesis officer.

$$\forall chair, \forall erep, \forall tofficer \square Do(chair, Return(erep, tofficer))$$

P16: The examining committee vote to get the final decision.

$$\forall ec, \forall thesis.status \square Do(ec, VoteDecide(thesis.status))$$

P17.1: When a thesis is accepted with major modification, the examining committee must include a precise description of the modifications in their report.

$$\exists thesis, thesis.status = \text{"accepted with major modification"} \rightarrow$$

$$\forall thesis.ec, \exists erep.modification \square Do(thesis.ec, AddToReport(erep.modification))$$

P17.2: When a thesis is accepted with major modification, the examining committee must also include a date of modification completion.

$$\exists thesis, thesis.status = \text{"accepted with major modification"} \rightarrow$$

$$\forall thesis.ec, \exists thesis.ec.erep.modi\_comp\_date$$

$$\square Do(thesis.ec, AddToReport(thesis.ec.erep.modi\_comp\_date))$$

P17.3: the candidate should modify the thesis.

$$\forall stud, \exists modification, \forall thesis \square Do(stud, ModifyThesis(thesis, modification))$$

P17.4: When a thesis is accepted with major modification, the candidate's supervisor should demonstrate to the examining committee that the required modification have been made.

$$\exists thesis, thesis.status = \text{"accepted with major modification"} \rightarrow$$

$$\forall sup, \forall thesis.ec, thesis \square Do(sup, Demo(thesis, thesis.ec))$$

P17.5: The examining committee should give an opinion about the modification.

$$\forall ec. \exists modification \square Do(ec, ApproveModification(modification))$$

P17.61: The candidate's supervisor should inform the thesis officer in writing that the modification has been approved.

$$\forall sup, \forall tofficer, ec.modi\_result \square Do(sup, Inform(tofficer, ec.modi\_result))$$

P17.62: If the modifications have been approved, the candidate's supervisor should write to the thesis officer about the result.

$$\forall ec.thesis, \forall ec \ ec.modi\_result = TRUE \rightarrow$$

$$\forall sup, \forall tofficer, ec.thesis.ec.modi\_result$$

$$\square Do(sup, Inform(tofficer, ec.thesis.ec.modi\_result))$$

P17.63: If the modification has not been approved, the candidate should continue to modify the thesis.

$$\forall ec.thesis, \forall ec \ ec.modi\_result = FALSE \rightarrow$$

$$\forall stud, \exists modification, \forall ec.thesis$$

$$\square Do(stud, ModifyThesis(ec.thesis, modification))$$

P18.1: If the examining committee is not prepared to reach a decision concerning the thesis at the time of the thesis defence, the chair should determine what additional information is required by the committee to reach a decision.

$$\forall ec.thesis, \forall ec$$

$$\neg Do_{ec.thesis.defence.defence\_begin\_time < t < ec.thesis.adjourn.adjourn\_end\_time}$$

$$(ec, VoteDeside(ec.thesis)) \rightarrow$$

$$\forall ec.chair, \forall ec.info \square Do(ec.chair, DeterAddiRep(ec.info))$$

P18.2: If the examining committee is not prepared to reach a decision concerning the thesis at the time of the thesis defence, the chair should arrange to obtain this information for the committee.

$\forall ec.thesis, \forall ec$

$\neg Do_{ec.thesis.defence.defence\_begin\_time < t < ec.thesis.adjourn.adjourn\_end\_time}$

$(ec. VoteDeside(ec.thesis)) \rightarrow$

$\forall ec.chair, \forall ec.info \square Do(ec.chair, ArrangeObtain(ec.info))$

P18.3: As soon as the additional information is available, the chair shall call another meeting of the committee.

$\forall ec ec.info \neq NULL \rightarrow$

$\forall ec.chair \square Do(ec.chair, CallMeeting())$

P18.4: The chair should inform the candidate that the decision is pending.

$\forall chair, \forall stud, \forall thesis.status \square Do(chair, Inform(stud, thesis.status))$

### 5.1.7 Step7: Develop a Mapping from RNL to the FR

*Develop a mapping from RNL to the FR to suit the further refinement or enforcement steps.*

It is hard to directly map the RNL to the FR. However, by proposing a Hidden Template (HT) as an intermediate representation between the two languages, we create a bridge and the translation process becomes much easier. In the new translation process, we first map the RNL into a HT and then map a HT into a FR.

#### 5.1.7.1 Hidden Template(HT)

We borrow the idea of Hidden Template from case grammar [7], [15]. In case grammar, the focus of analyzing a sentence is turned into its conceptual structure (deep case) rather than to its syntactic construction. Taking the verb as the key, the sentence actually describes one or more semantic relationships between the verb and its “instruments”. Following this idea, we created a Hidden Template which contains several logically related items and centers on an action item (shown in Table 5.2). Once we fill the information contained

in a RNL sentence into this HT, we can obtain this sentence's semantic meaning in a straightforward way.

Item of HT	RNL Group	Semantic Meaning
0. Policy No.:	P15.1	
1. Subject Name:	examiner	ecm
2. Subject Scope:	each	$\forall$
3. Target1 Name:	evaluation form	eform
4. Target1 Scope:	the	$\forall$
5. Target2 Name:		
6. Target2 Scope:		
7. Complement Name:		
8. Complement Scope:		
9. Action:	sign	ecm.Sign(Doc)
10. Action Type:	must	$\square$
11. Constraint:		
12. Trigger Event:		

Table 5.2: The Hidden Template for the policy “*Each examiner must sign the evaluation form*”

In Table 5.2, column 1 describes the structure of the table; column 2 records the corresponding RNL groups of the sample policy statement; column 3 shows the real content of the sample policy's HT. The meaning of items in the HT are explained below:

**0. Policy No** records which policy this HT describes.

**1. Subject Name** records the entity's name which acts as the agent of the action.

**2. Subject Scope** records the quantity of the subject. Two quantifiers can be used,  $\forall$  and  $\exists$ .

**3. Target1 Name** is the entity's name who acts as the target of the action.

**4. Target1 Scope** is the same as Subject Scope.

**5. Target2 Name** is reserved for the actions which have two targets. For example, the action “submit” has two targets: a document and a person.

**6. Target2 Scope** is the same as Subject Scope.



- 7. Complement Name** is reserved for the target that does not appear in the RNL sentence. However, according to the semantic meaning of the verb, it should be added to the HT. For example, in the policy *“the thesis office may request a postponement to the members of the examining committee”*, the verbnome *“request a postponement”* has the semantic meaning *“TOffice.RequestPostpone(ECm, Thesis)”*. Although the concept “Thesis” in the above semantic meaning does not appear in the surface level policy, it is an essential part of the action “RequestPostpone”. For this reason, the entity *“thesis”* is added into the “Complement Name” item.
- 8. Complement Scope** is the same as Subject Scope.
- 9. Action** records the action of the policy sentence.
- 10. Action Type** records the obligation/authorization information of the action.
- 11. Constraint** records various of constraints of the action.
- 12. Trigger Event** records the trigger condition of the policy.

#### 5.1.7.2 Map the Policies in RNL to the HT Formats

Based on the eight grammar rules listed in Figure 5.6, Table 5.3 shows the relationship between a RNL sentence and its HT. In Table 5.3, column 1 lists the structural component of the HT and column 2 enumerates the mapping relationship between that HT item and the surface sentence structure.

According to the mapping relationship listed in Table 5.3, we can directly map the appropriate contents in the RNL sentence into the right HT rows without ambiguity. These HT rows include **Subject Name/Scope**, **Target1 Name/Scope**, **Target2 Name/Scope**, **Action** and **Action Type**.

We notice that the AG (Adjective/Adverb group) falls into two possible rows: “Constraint” and “Trigger Event”, which means AG could be either a constraint or a trigger event. We use the first word of AG to decide if it should be filled into the “Constraint” row

HT Component	Mapping Relationship (HT $\iff$ RNL Syntax)
Policy No.:	
Subject Name:	NG1 $\rightarrow$ NOUNNAME/NOUN
Subject Scope:	NG1 $\rightarrow$ DETER
Target1 Name:	NG2 $\rightarrow$ NOUNNAME/NOUN
Target1 Scope:	NG2 $\rightarrow$ DETER
Target2 Name:	NG3 $\rightarrow$ NOUNNAME/NOUN
Target2 Scope:	NG3 $\rightarrow$ DETER
Complement Name:	
Complement Scope:	
Action:	VG $\rightarrow$ VERBNAME/VERB
Action Type:	VG $\rightarrow$ AUX
Constraint:	AG
Trigger Event:	AG

Table 5.3: The Mapping Relationship between the HT and RNL's Syntax

or the "Trigger Event" row. Let's see an example. Suppose we have a policy "*The examiners should submit the written report to the chair before the thesis defence begins.*". The AG in this sentence is "*before the thesis defence begins*". We check the semantic meaning of the word "*before*" in the vocabulary and find match shown in Figure 5.8.

The word "*before*" has a "{**Constraint**}" sign in its semantic meaning column, so we fill this AG into "Constraint" row. If the word has a "{**Trigger**}" sign (see the word "*in*" in Figure 5.8 as an example), we will fill the AG into "Trigger Event" row. According to the AG grammar shown in Section 5.1.4.2, only two categories of words can be lead-ins of an AG. "PREP" and "CONJ". Since words in category "PREP" and "CONJ" are all closed words which have unique semantic meanings, we won't have ambiguity problems and misplace the AGs.

We also notice that the **Complement Name/Scope**'s mapping relationship items are empty, which means no content in the RNL sentence can be directly filled into these two items. This is easy to understand. Assume an action mentioned in a RNL format policy needs two object parameters. Only one is mentioned in the policy sentence. We still need the other one to complete this action. In this case, we fill the object parameter that doesn't

Seg	SegmentName	SegmentTyp	Segme	SegmentMeaning
64	when	conj	1	{trigger}
142	as soon as	conj	3	{trigger}
29	if	conj	1	{trigger}
24	for	prep	1	{trigger}
15	in	prep	1	{trigger}
1	upon	prep	1	{trigger}
49	before	prep	1	{constraint} t<
507	after	prep	1	{constraint} t>

Figure 5.8: The Sementic Meaning of “before” in the Vocabulary

appear in the RNL format policy into **Complement Name/Scope** items. Let’s see an example:

**Example:** “*The thesis office may request a postponement addressed to the member of the examining committee.*”

The structural analysis of this sentence is shown in Figure 5.9.

The action “*request a postponement to*” has the match shown in Figure 5.10 in the vocabulary.

We notice that method *RequestPostpone* has two parameters *ECm* and *Thesis*. The parameter *ECm* in this sentence is filled by NG2 “*the members of the examining committee*”. The parameter *Thesis* isn’t mentioned in the sentence. However, the concept of “thesis” is a necessary part to complete the action *RequestPostpone*. For this reason, we fill “thesis” and “V” (default setting) into the **Complement Name/Scope** items to make the sentence concepts complete.

### 5.1.7.3 Map the HT to the FR

The mapping relationship from HT to the FR is quite straightforward. The following BNF captures the essence of how to obtain FR format’s policy representation from its corresponding HT representation.

$$(1) \langle Policy \rangle ::= \langle TriggerEvent \rangle \rightarrow \langle Policy \rangle$$

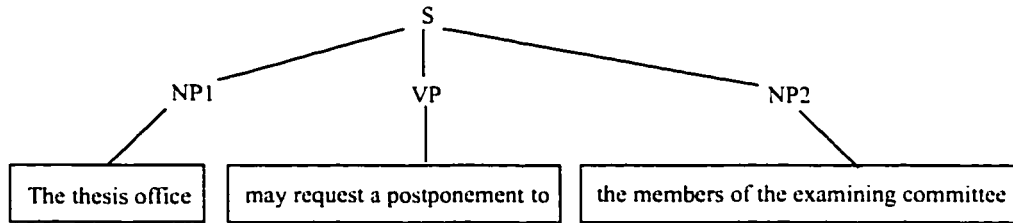


Figure 5.9: The Semantic Meaning of “request a postponement to” in the Vocabulary

$$(2) \langle Policy \rangle ::= \langle SubjectScope \rangle \langle SubjectName \rangle \langle ComplementScope \rangle \\ \langle ComplementName \rangle \langle Target1Scope \rangle \langle Target1Name \rangle \\ \langle Target2Scope \rangle \langle Target2Name \rangle \\ \langle ActionType \rangle_{Constraint} Do(\langle SubjectName \rangle, \langle Action \rangle)$$

$$(3) \langle TriggerEvent \rangle ::= \langle Policy \rangle$$

$$(4) \langle SubjectScope \rangle ::= \forall | \exists$$

$$(5) \langle Target1Scope \rangle ::= \forall | \exists$$

$$(6) \langle Target2Scope \rangle ::= \forall | \exists$$

$$(7) \langle ComplementScope \rangle ::= \forall | \exists$$

$$(8) \langle ActionType \rangle ::= \diamond | \square$$

(9) The content of  $\langle SubjectName \rangle$ ,  $\langle Target1Name \rangle$ ,  $\langle Target2Name \rangle$ ,  $\langle ComplementName \rangle$ ,  $\langle Action \rangle$  and  $\langle Constraint \rangle$  are coming from the vocabulary.

The following example explains how the above formulas work. Let’s see the same example we used in Table 5.2 in Section 5.1.7.1:

**Example:** “Each examiner must sign the evaluation form.”

Se	SegmentName	SegmentT	Se	SegmentMeaning
32	request a postponement to	verbrname	4	[Toffice,Postpone,RequestPostpone,ECm,Thesis]

Figure 5.10: The Semantic Meaning of “*request a postponement to*” in the Vocabulary

The HT is already listed in Table 5.2 in Section 5.1.7.1. According to the formulas listed above, a FR format of the policy is generated as follows:

$$\forall ecm, \forall eform \square Do(ecm, Sign(eform))$$

At this stage, by following the methodology presented in Chapter 4, we have constructed the RNL of the policies of the VTO. The RNL includes:

- an object-oriented model describing the domain knowledge.
- a vocabulary recording the words and segments used in the RNL.
- eight grammar rules used to construct legal RNL sentences.
- a formal logic-based representation (FR) used to represent the policies inside the software system, and
- a mapping technique to map the RNL format policies to the FR format policies.

## 5.2 Virtual Call Center (VCC)

A Virtual Call Center (VCC) simulates a real call center center which provides technical support to the customers by answering the incoming calls.

In the call center scenario, the customer can contact employees in VCC by telephone. One of the employees takes the call, identifies the problem and assigns it to one of the available experts. The expert might contact the customer for more information, or solve the problem, or redirect the call to other expert.

### 5.2.1 Step1: Define the Problem Domain

*Define the problem domain by capturing “adequate” number of use case scenarios and creating a sample set of policy statements. The use case maps can be drawn at this step in order to visually representing the causal use case scenarios combined with structures.*

#### 5.2.1.1 Use Case Maps

To define the problem domain of the VCC, we draw a use case map (Shown in Figure 5.11) describing the call center scenario.

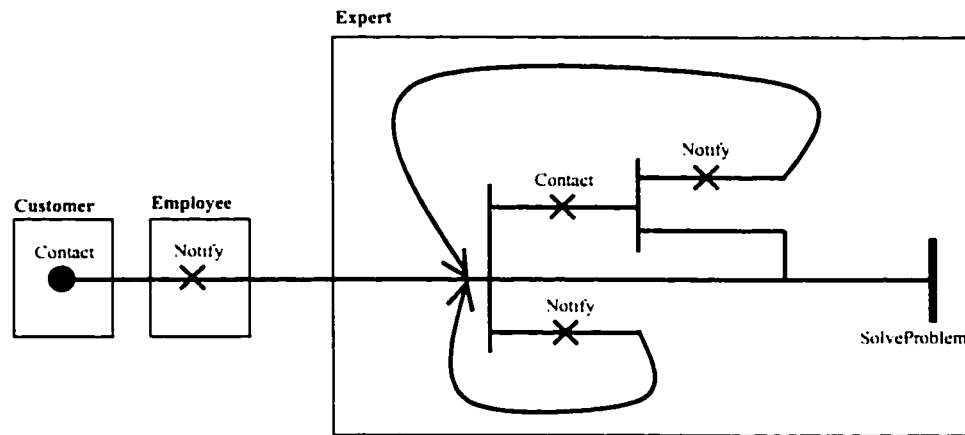


Figure 5.11: The Use Case Map for the VCC system

In Figure 5.11.

[*Customer, Employee and Expert*] are three object names.

[*Contact in object Customer*] means “the customer contacts the employees in VCC by phone”.

[*Notify in object Employee*] means “the employee takes the call. identifies the problem and assigns it to one of the available experts”.

[*Contact in object Expert*] means “the expert contacts the customer for more information”.

[*Notify in object Expert*] means “the expert redirects the problem to another expert”.

[*SolveProblem in object Expert*] means “the expert solves the problem”.

#### 5.2.1.2 Sample Policies

A set of sample policies is set up at the same time.

P1 The customer may contact employees in VCC for technical support.

P2 The employee in VCC should notify an expert about the problem.

P3 The expert may solve the problem.

P4 The expert may contact the customer for more information.

P5 The expert may redirect the call to another expert.

#### 5.2.2 Step2: Create the Object-Oriented Model

*Create an object-oriented model for the problem domain. In this stage, the objects are clustered into classes; the attributes and the methods of classes are defined; the relationships and interactions between classes are determined.*

The object-oriented model for the call center scenario is created in Figure 5.12. In Figure 5.12, *Person* is an abstract class generalized from *Employee* and *Customer*. *Expert* is a subclass of *Employee*. An instance of *Person* can send messages to other *Person* objects by using the method *Contact*. An instance of *Employee* can inform other *Employee* objects about the incoming call by using the method *Notify*. An instance of *Expert* can answer the incoming call and solve the customer’s problem by using the method *SolveProblem*.

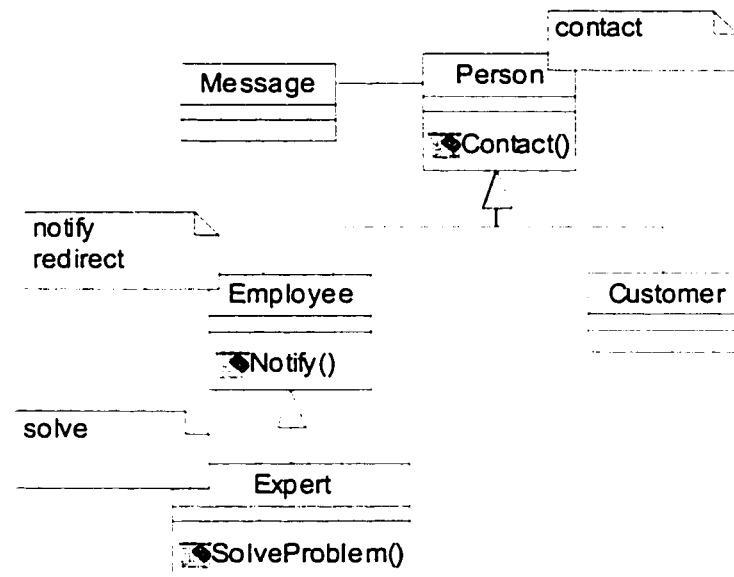


Figure 5.12: Class Diagram for the VCC Model and the Verbs Listed in the Vocabulary



### 5.2.3 Step3: Construct a Vocabulary

*Based on the cases selected in the problem domain and the domain expertise, construct a vocabulary.*

The verbs listed in the vocabulary can be seen in Figure 5.12. Figure 5.13 lists the complete vocabulary.

### 5.2.4 Step4: Construct the Syntax

*Select a minimal set of simple-to-use syntactic structures for the intended RNL.*

The syntax of VCC system is shown in Figure 5.14. The grammar for NG and VG is the same as those we used in the VTO system. We observe that the grammar rules we used in the VCC system is only a subset of those we used in the VTO system.

### 5.2.5 Step5: Manually Verify the RNL's Habitability

*Verify by manual methods to see if all the use case scenarios can be expressed in RNL, that is to verify if the RNL satisfies the "habitability" requirement.*

We manually rewrote the sample policies of the VCC system using the vocabulary constructed and the grammar rules selected. We found that the vocabulary and the grammar are large enough to express everything happened in the call center scenario. Therefore, the RNL of VCC system is 100% habitable.

### 5.2.6 Step6: Develop a Formal Logic-based Policy Specification

*Develop a formal logical specification based on the semantics created by the object-oriented model.*

We can use the same formal representation (FR) developed in the VTO system. The FRs for the five sample policies are listed as follows.

Se	SegmentName	SegmentT	S	SegmentMeaning
1	the	deter	1	{for all}
2	customer	noun	1	customer
3	can	aux	1	{must}
4	contact	verb	1	[Person,bool,Contact,Person,Message]
5	employee	noun	1	employee
6	employee in VHDC	nounname	4	employee
7	should	aux	1	{must}
8	notify	verb	1	[Employee,bool,Notify,Person,Message]
9	an	deter	1	{exist}
10	expert	noun	1	employee
11	may	aux	1	{may}
12	solve	verb	1	[Eepert,bool,SolveProblem,Message]
13	problem	noun	1	message
14	call	noun	1	message
15	for	prep_subj	1	
16	information	nounname	1	message
17	redirect	verb	1	[Employee,bool,Notify,Person,Message]
18	to	prep_subj	1	
19	another	deter	1	{exist}
20	expert	noun	1	expert
21	about	prep_subj	1	
22	more	deter	1	{exist}
23	technical support	nounname	2	message

Figure 5.13: The Vocabulary for the VCC system

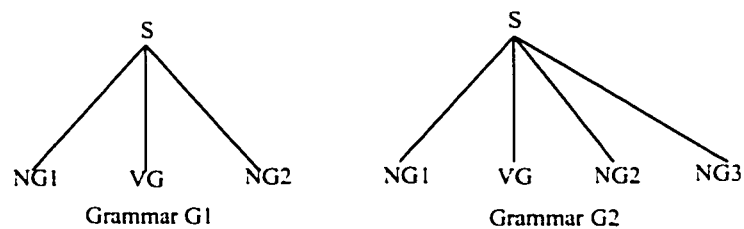


Figure 5.14: The Grammar Used in VCC system

- (1) The customer may contact employees in VCC for technical support.

$$\forall customer, \exists employee, \exists message \diamond Do(customer, Contact(employee, message))$$

- (2) The employee in VCC should notify an expert about the problem.

$$\forall employee, \exists expert, \exists message \square Do(employee, Notify(expert, message))$$

- (3) The expert may solve the problem.

$$\forall expert, \exists message \diamond Do(expert, SolveProblem(message))$$

- (4) The expert may contact the customer for more information.

$$\forall expert, \forall customer, \exists message \diamond Do(expert, Contact(customer, message))$$

- (5) The expert may redirect the call to another expert.

$$\forall expert, \exists expert1, \exists message \diamond Do(expert, Contact(expert1, message))$$

### 5.2.7 Step7: Develop a Mapping from RNL to the FR

*Develop a mapping from RNL to the FR to suit the further refinement or enforcement steps.*

Since the syntax of the RNL we developed for the VCC system is only a subset of the one we used in the VTO system, and since we use the same FR as the one in the VTO system, we adopt the same mapping technique used in the VTO system without modifying anything.

At this stage, following the methodology we presented in Chapter 4, a new RNL for the VCC system is developed. It includes:

- an object-oriented model describing the domain knowledge.
- a vocabulary recording the words and segments used in the RNL.
- two grammar rules used to construct legal RNL sentences.

- a formal logic-based representation (FR) used to represent the policies inside the software system, and
- a mapping technique to map the RNL format policies to the FR format policies.

### 5.3 Summary

In this chapter, we presented two examples to demonstrate how to construct a RNL for a policy-based system. Using the methodology we presented in Chapter 4, we can construct a RNL for any policy-based system as long as the policies satisfy the conditions we mentioned in Section 4.1.

- The policies are changed dynamically and frequently.
- They are simple policies.
- They are suitable to be described in natural language.

We also notice that some parts of the RNL are quite stable and seldom change when the problem domain changes from one to another. These parts include:

- the syntax,
- the FR and
- the mapping technique used to map the policy from its RNL format to the FR format.

Once we create a RNL whose syntax is sufficiently large to contain every possible grammar rule, we can retain the syntax, the FR and the mapping technique of this RNL as reusable components. When we create a new RNL for a new problem domain, we can just develop the object-oriented model and the vocabulary, and copy the reusable parts to this new RNL. It will save time and resources. This reusability is demonstrated in constructing the RNL for the VCC system.

In the next chapter, we will present a prototype RNL interface for the VTO system to test the methodology we explained in this chapter. The VTO system is a policy-based system that handles the procedures related to the submissions and the examinations of the students' theses. A RNL interface for this VTO system can accept the end user's RNL policies as inputs, analyze them automatically, translate them into FR format, and save them into the policy repository.

## Chapter 6

### VTO System Design and Implementation

#### 6.1 Implementation Environment

A prototype called VTO (Virtual Thesis Office) System is designed and implemented for testing the methodology we proposed in this thesis. The VTO system is a policy management tool for the domain of the Master's thesis examination in Concordia University. It accepts the RNL format of policies which are restricted in the problem domain, analyzes the RNL format inputs, translates them into the FR format policies and saves them into a policy repository. The users of the VTO system can review the policy repository, add new policies into it, update existing policies, and delete policies from the repository.

The VTO system has been implemented using the JAVA programming language, combined with a Microsoft Access database to store the knowledge base and the policies. A JDBC-ODBC driver is used to access the Microsoft Access database. Following is a list of software packages we used to develop the VTO system:

- JDK Standard Edition 1.2
- JDBC-ODBC Driver 1.1
- Microsoft Access 97

#### 6.2 Software Architecture

The basic architecture of the proposed system can be subdivided into two major components: the policy management tool and the RNL processor. The policy management tool

provides an interface to add, update and delete policies into or from the policy repository. The RNL processor analyzes input policies stated in the RNL format, translates the RNL policy into the FR format and saves both the RNL format and the FR format in the policy repository. The overall architecture is shown in Figure 6.1.

In the following sections, we will briefly introduce the functions of each module and the algorithms we used.

### 6.3 Policy Management Tool

The policy management tool (Figure 6.2) provides to end users an interface to add, update, delete policies into or from the policy repository.

#### 6.3.1 The Time Sequence of the Policies

Generally, a time sequence of actions exists in the set of policies of VTO. In a policy-based system, the policies being executed earlier may affect the system states, and some of these system states may become a part of the conditions considered in the subsequently applicable policies. This kind of condition relationship between early-executed and later-executed policies can hardly be expressed in a single policy sentence. Alternatively, a sequence of policy procedure such as “step1, step2, ... .. step10... ..” becomes an appropriate way to record this time sequence.

##### 6.3.1.1 The Time Scope of the Policies

Before we introduce the time sequence of the policies, we first introduce a time scope concept.

Each policy, no matter the obligation type or the authorization type, has an effective time scope. In Figure 6.3, Policy P1 is active in P1’s time scope. Outside of that scope, policy P1 has no effect on the system. An example follows:

**Example 1:** *The chair will give priority to questions from members of the examining*

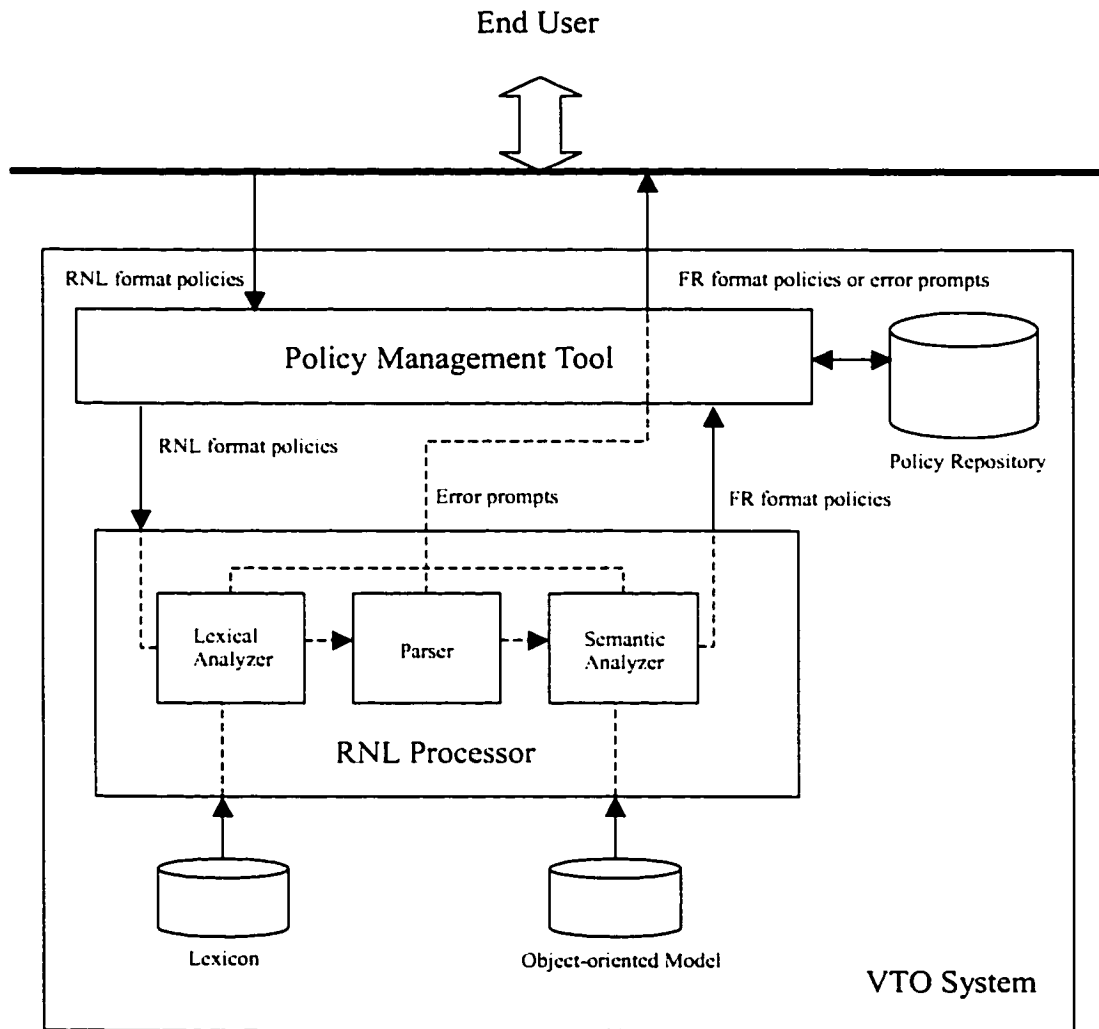


Figure 6.1: The VTO system's Architecture



**Policy Management** [Icons]

File View Record Help

[Icons]

Current Policy: 1 Row 1 is now selected.

Policy No	Policy: RNL Form Input	Policy: Formal Form Output
1	upon receipt of the thesis from the...	{exist}tofficer, {exist}stud Do...
1.2	upon receipt of the thesis from the...	{exist}tofficer, {exist}stud Do...
12	the chair will give priority to questi...	{for all}chair, {for all}question ...
13	only appointed members of the ex...	{for all}ecm, {for all}thesis.deli...
14.2	the chair should check the examini...	{for all}chair, {for all}erep{mu...
15.1	each examiner must sign the exa...	{for all}ecm, {for all}eform{mu...
15.2	each examiner must sign the exa...	{for all}ecm, {for all}erep{mus...
15.3	the chair must return the examiner...	{for all}chair, {for all}eform, {f...
15.4	the chair must return the examinati...	{for all}chair, {for all}erep, {for...
16	the examining committee vote to g...	{for all}ec, {for all}thesis.statu...
17.1	when a thesis is accepted with m...	{exist}thesis, thesis.status="a...
17.2	when a thesis is accepted with m...	{exist}thesis, thesis.status="a...
17.3	the candidate should modify the th...	{for all}stud, {exist}modificatio...
17.4	when a thesis is accepted with m...	{exist}thesis, thesis.status="a...
17.5	the examining committee should gi...	{for all}ec, {exist}modification...
17.61	the candidate's supervisor should	{for all}sun {for all}tofficer ec

**Policy Card**

**Policy No: 1**

**Restricted Natural Language Expression of the Policy:**  
*upon receipt of the thesis from the thesis office, the graduate committee*

**Formal Form Expression of the Policy:**  
 $\exists tofficer, \exists stud Do(stud, Submit(tofficer, stud.thesis)) \rightarrow \forall stud.gsc, \forall stud$

Figure 6.2: The VTO System's Policy Management Tool

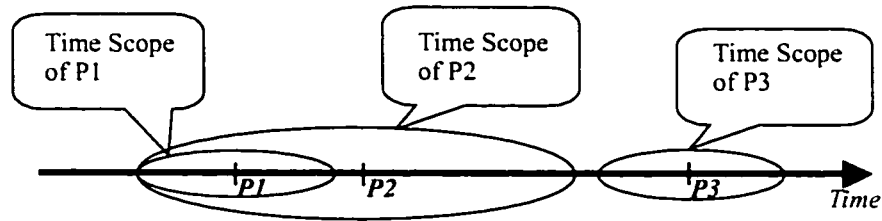


Figure 6.3: The Time Scope of Policies

*committee during the thesis defence.*

It is obviously that policy P10 given above is effective after the thesis Defence has begun and before the defence finish. Outside this period of time, this policy is not meaningful.

Time scopes exist in not only obligation policies but also in authorization policies. Consider the following example:

**Example 2:** *Any member of the university is free to attend a master's oral thesis defence.*

This is an authorization policy. The member of the university can attend the thesis defence only after the defence has begun and before the defence finishes. In other time this policy has no effect.

### 6.3.1.2 The Time Sequence of the Policies

The policy's time scope starts at a *begin\_time* and ends at an *end\_time*. Assume we have two policies *P1* and *P2*. If

$$begin\_time_{p1} \leq begin\_time_{p2} \cap end\_time_{p1} \leq end\_time_{p2}$$

then we say *P1* should be executed earlier than *P2*. If we sort the policy set  $\{P1, P2\}$  in an ascending order according to the time sequence, *P1* should be in a earlier place than *P2*.

Figure 6.3 is an example demonstrating the time sequence in a policy set  $\{P1, P2, P3\}$ .

Since

$$begin\_time_{p1} \leq begin\_time_{p2} \leq begin\_time_{p3} \cap end\_time_{p1} \leq end\_time_{p2} \leq end\_time_{p3}.$$

$P1$  should be executed earlier than  $P2$  and  $P2$  should be executed earlier than  $P3$ . The time sequence of this policy set is

$$P1, P2, P3$$

In our policy management tool, we use a Policy No. to order the time sequence of the policy set. The policies having smaller Policy No.s should be executed earlier. For example, assume we have four policies labeled as  $P1, P1.1, P1.2, P2$ , then the time sequence of this policy set is

$$P1, P1.1, P1.2, P2$$

$P1$  should be executed earlier than  $P1.1, P1.2$  and  $P2$ . Same way,  $P1.1$  should be executed earlier than  $P1.2$  and  $P2$  but later than  $P1$ .

The reason we order the policy set in the policy repository is to keep our translation from RNL to FR simple. As we mentioned above, every policy should have a time scope, then each policy in RNL format should have phrases to describe the *begin\_time* and *end\_time* of the time scope since we analyze the policy based on a single policy sentence. However, this is not the way people communicate in a natural language. In our VTO system, almost every RNL policy omits one or both the time scope boundaries in its policy sentence. Instead of using phrases to specify the time scope, our RNL implies a policy time sequence by specifying the policies in an execution time order. In order not to lose this information and to avoid analyzing time scope phrases in every policy sentence, we use "Policy No." to record this information and let the end users decide what "Policy No." an input policy should have.

## 6.4 RNL Processor

The RNL Processor is combined with three components: (Shown in Figure 6.1)

- (1) a lexical Analyzer.
- (2) a parser, and
- (3) a semantic analyzer.

### 6.4.1 Lexical Analyzer

The lexical analyzer accepts the input RNL sentence from the policy management tool. It tokenizes the sentence into words, compares each word with the one in the lexicon, corrects the spelling errors, and generates a segment list for the parser.

#### 6.4.1.1 The Data Structure of the Lexicon

Our VTO system keeps two database tables to save the lexical information. They are “**Word**” table and “**Segment**” table. As we mentioned in Chapter 5, every word or word group which has a semantic meaning recorded in our lexicon is called a **segment**. We record these segments into our “Segment” table. Along with the semantic meaning of each segment, we also save the information about how the segment looks like, with which words it combines, how many words it has, what category it belongs to etc. Similarly, we record the information about how to spell a single word and in which position this word should be in a segment into the table “Word”. In the next section, we will explain how these two tables are used in the lexical analysis.

#### 6.4.1.2 The Definition of Similar Words

There is an important concept in our lexical analysis, the **similar words**. The similar words are those words similar in spelling. To provide similar words for the original word is a function the Lexical Analyzer provided with, when it does the spelling check of the input

sentence. If the user inputs some words not existed in our lexicon, the lexical analyzer will give him/her a similar word list, help him/her to correct the spelling errors and choose the words the system known. For this purpose, a definition of **similar words** is needed. We adapt Winiwarter's similarity measurement approach [46] to measure to what degree two words are similar. After introducing this similarity measurement approach, we will give the definition of **similar words**.

Assume we have two words  $w1$  and  $w2$ .  $w1$  and  $w2$ 's similarity degree can be measured by the following formula:

$$SIM(w1, w2) = 1 - \frac{\sum_{i=1}^k |c_{1,i} - c_{2,i}|}{\sum_{i=1}^k c_{1,i} + \sum_{i=1}^k c_{2,i}}$$

Here,  $w1$  is the initial user input word,  $w2$  is the word in the word dictionary,  $c_{1,i}(c_{2,i})$  signifies the number of occurrences of character  $i$  in the first (second) term  $w1(w2)$ ,  $k$  is the sum of different character numbers in  $w1$  and  $w2$ .

A suitable threshold  $p$  can be chosen. The terms whose similarity degree is larger than  $p$  are treated similar to the original word.

This similarity measure is not always right. Consider the following pathological case:

if  $w1 = \text{"ommunicationc"}, w2 = \text{"communication"}$ ,

then  $SIM(w1, w2) = 1$

This result is certainly wrong because  $w1$  has an obvious error. However, for most of the practical situations, this method leads to a reasonable result. That's why we adopt it as our spelling similarity measure.

Now suppose

$w1$  is the initial input word,

$w2$  is a word in the word dictionary,

$len\_w1$  is the length of  $w1$ ,

$len\_w2$  is the length of  $w2$ ,

$p$  is the pre-defined threshold (for example 80%) of the similarity measure  $SIM$ .

we define **the two words are similar** if and only if

$$(3 \leq len\_w1 \leq 11) \wedge (|len\_w1 - len\_w2| \leq 1) \wedge (SIM(w1, w2) \geq p)$$

or

$$(12 \leq len\_w1) \wedge (|len\_w1 - len\_w2| \leq 2) \wedge (SIM(w1, w2) \geq p)$$

### 6.4.1.3 Two-Phase Lexical Analysis Algorithm

Our lexical analyzer uses a two-phase lexical analysis algorithm (shown in Figure 6.4 to check the spelling errors and construct a segment list for the following analysis).

- **Phase one:** A spelling check is done. If the word has spelling errors, a list of similar words is suggested to the end user to correct the errors.
  - (1) Tokenize the input sentence into single words.
  - (2) For each individual word, look up the “Word” table to find the match. If the match is not found, then use spelling similarity measurement algorithm (described in the next section) to check every possible **similar word** in table “Word” to find out the most similar one (or several words). If one or several words’ similarity degree  $\geq p$  (say 80%), then use them as possible similar words and ask the end user to correct the spelling error.
  - (3) The output of phase one is a list of correctly spelled words.
- **Phase two:** Browse the word list generated in phase one and match the segments by looking up the “Segment” table. The output of phase two is a segment list which records the segments’ types and their semantic meanings. The information about segments’ types will be used for parsing. Their semantic meanings will be analyzed in semantic analysis.

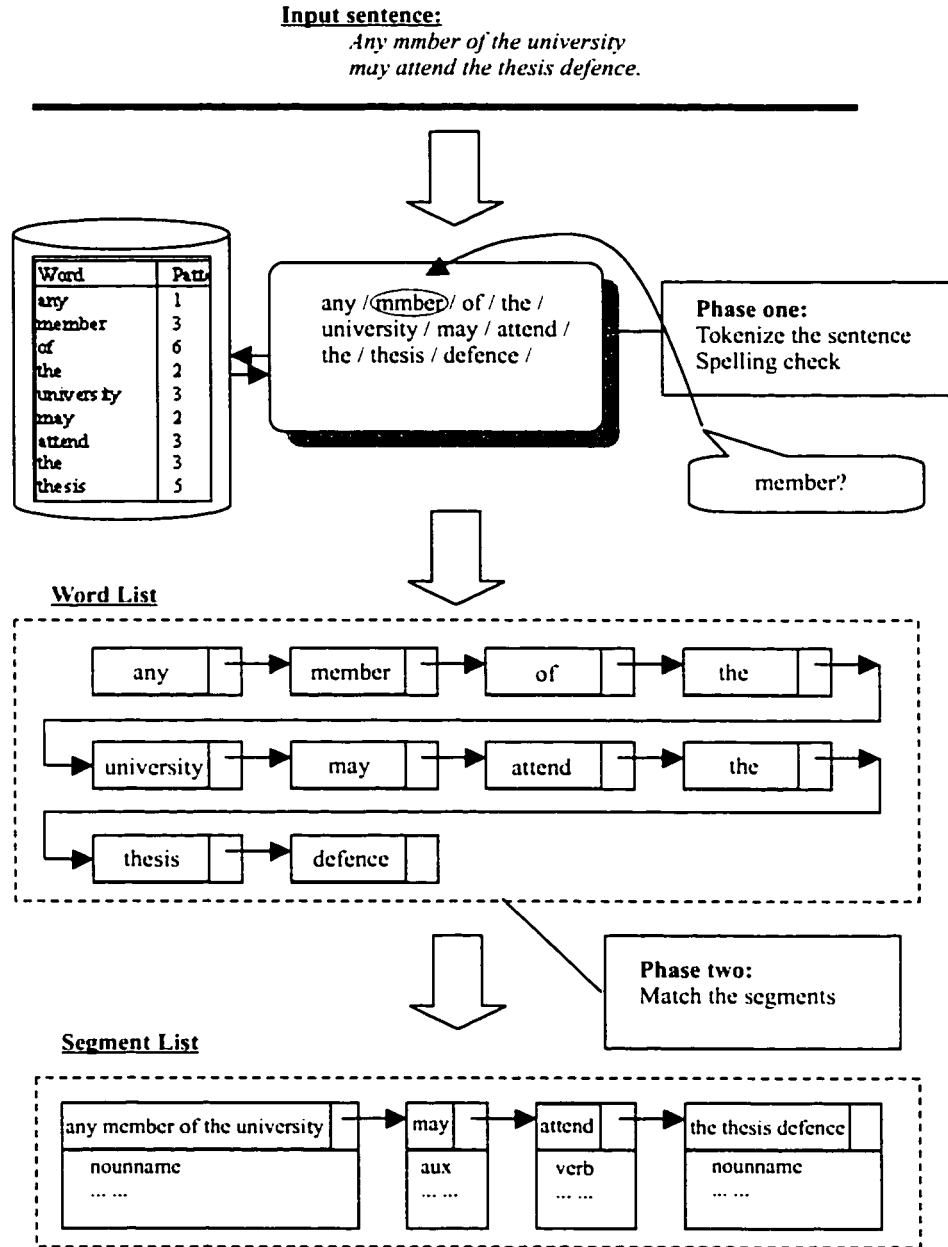


Figure 6.4: The Process of Lexical Analysis

Note that a segment may belong to several segment categories or have several semantic meanings. This phenomenon is called ambiguity. At this stage, we record every category and semantic meaning into the segment list and submit them to a parser. The parser and the semantic analyzer will check every possible combination to reject the inappropriate segment categories or semantic meanings.

#### 6.4.2 Parser

The parser takes the segment list generated by the lexical analyzer as input, analyzes the sentence's syntactical structure and generates a parse tree for further semantic analysis.

In the VTO system, we use a LALR parser generator "CUP" [21] to generate our LALR parser. Figure 6.5 is an example of the parse tree the parser generated.

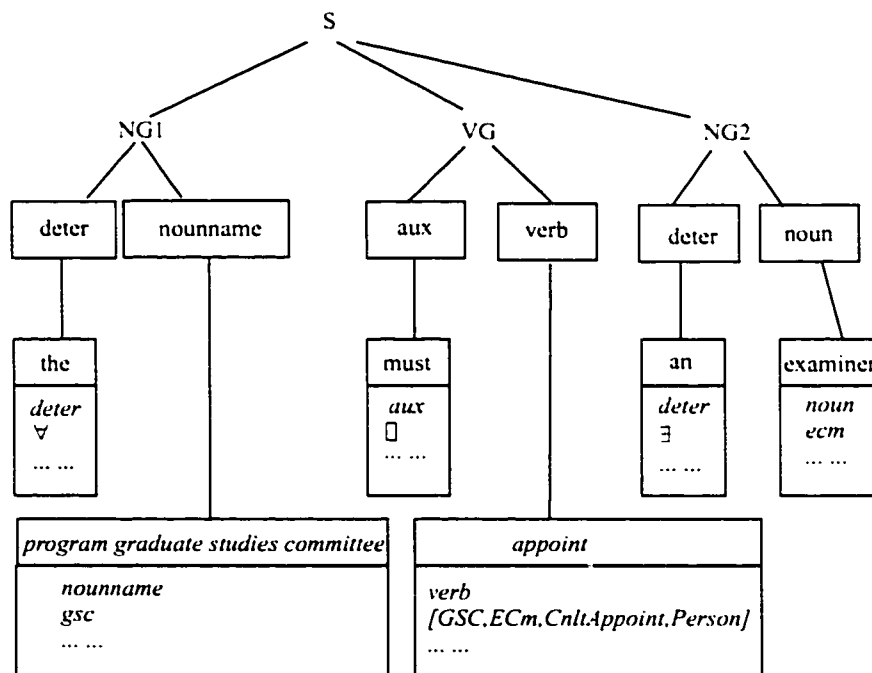


Figure 6.5: A Parse Tree Generated from the Parser

Note that if ambiguous segments exist in the input sentence, the parser may generate more than one parse tree, all of which are syntactically correct. In this situation, the ambiguity can only be resolved in the semantic analysis stage by analyzing the semantic



meanings of different parse trees.

### 6.4.3 Semantic Analyzer

Semantic analyzer analyzes the content of the policy sentence, resolves the ambiguity problems, and generates the policy in FR format. Our semantic analyzer divides the semantic analysis into three steps.

Step1: For each parse tree, fill each segment's semantic meaning into the Hidden Template(HT).

Step2: Do the consistency check according to the object-oriented model.

Step3: If the sentence passes the consistency check, then generate the policy in FR format. If not, the semantic analysis has failed and the input sentence contains illegal contents and cannot be accepted by the RNL Processor.

#### 6.4.3.1 Fill the Hidden Template

Filling the HT is the means that the VTO system uses to "understand" the RNL input sentence. In Table 5.3, a relationship between the RNL's syntax and the HT has been established. We can follow this relationship to fill the appropriate segment's semantic meaning in the parse tree to the HT. For example, for the parse tree shown in Figure 6.5, a HT shown in Table 6.1 is filled.

Note that one segment may have a possibility to have multiple meanings. For example, in our VTO system, verb "inform" may represent class *Chair*'s method

*[bool Inform(Stud, ThesisStatus)]*

(which means "the chair informs the student about the thesis status"); it may also represent class *Sup*'s method

*[bool Inform(Tofficer, ModiResult)]*

Policy No.:	P4	
Subject Name:	graduate studies committee	gsc
Subject Scope:	the	$\forall$
Target1 Name:	examiner	ecm
Target1 Scope:	an	$\exists$
Target2 Name:		
Target2 Scope:		
Complement Name:		
Complement Scope:		
Action:	appoint	GSC.CnltAppoint(Sup,Person)
Action Type:	must	<input type="checkbox"/>
Constraint:		
Trigger Event:		

Table 6.1: An Example of a HT

(which means “the candidate’s supervisor informs the thesis officer if the modification of the thesis is approved”). In this situation, the semantic analyzer should check

- (1) if the method’s owner matches the subject in the HT.
- (2) if the method’s parameters match the target in the HT.

Only the matched one can be chosen to fill the action item in the HT.

#### 6.4.3.2 Consistency Check

Consistency check is the main task of the semantic analysis. It will succeed if the translation of the input sentence is logically reasonable. In the VTO system, since we fill the translation of the input sentence in the HT, we make the following checks sequentially for the HT to see if the translation is logically reasonable.

- (1) Does the action’s owner match the subject in **Subject Name** item?
- (2) Does each parameter of the action match the target in **Target1 Name** or **Target2 Name** or the complement in **Complement Name** item?

If the answers of the above two questions are “yes” then we say the translation passes the consistency check and it is logically reasonable. If the translation doesn’t pass the

consistency check, it means either the subject cannot do the action described in the HT or the subject doesn't do the action the same way described in the HT. In both situations, the translation has some logical errors and should be rejected by the semantic analyzer.

### 6.4.3.3 Generate the Output

Once the translation passes the consistency check, a policy in FR format can be generated following the formulas introduced in Section 5.1.7.3.

### 6.4.4 Ambiguity Problem

In the VTO system, ambiguity problems exist in every stage of the RNL analysis. If in the lexicon, a segment has more than one segment types or more than one segment meanings, an ambiguity problem may raise when that segment is used in an input policy sentence. To avoid or solve this problem, we adapt several methods in the design the VTO system.

- (1) To simplify the problem, we try to avoid most ambiguity problems by separating the lexicon into two parts, the "closed word" part and the "open word" part (Chapter 4). In the "closed word" part, every segment has a unique type and meaning. They are defined by the designer and cannot be modified by end users. For this reason, they seldom cause an ambiguity problem. We include as most words as possible into the "closed word" part. These words include every word in segment type **deter**, **prep**, **prep\_subj**, **conj**, **aux** and **adv**. On the other hand, we have to allow users to add objects and actions to the existed object-oriented model in case they have the needs to expand the knowledge base(object-oriented model). As a result, new segments that represent the objects and actions are added into the existed lexicon at the same time. These segments are categorized into the "open word" part of the lexicon. In this part, a segment might have more than one segment type or segment meaning and may cause ambiguity problems. We restrict the "open word" part into

only those **nouns, nounnames, verbs and verbnames**. In this way, we restrict the ambiguity problems into a limited area of the lexicon.

- (2) We also try to use word groups instead of single words to represent objects and object's attributes. By adopting this method, we reduce the chances of noun/nounname segments having multiply meanings. For example, segment "committee" in our lexicon may represent "the graduate committee" or "the examining committee". If we replace the single word "committee" as two word groups "the graduate committee" and "the examining committee", the ambiguity problem is avoided.
- (3) If the system finds a segment which has more than one segment types in the process of syntactic analysis, it will try to analyze each type of the segment to generate the parse trees. In some situations, only one type of the segment is "correct" and can generate a parse tree. In that case, the ambiguity problem is solved in the syntactic analysis stage. In other situations, more than one parse trees can be generated. The additional semantic analysis has to be done to solve the ambiguity problems.
- (4) In the following situations, the ambiguity problems have to be solved in semantic analysis.
  - The system finds a segment which has multiple segment meanings.
  - The syntactical analyzer generates more than one parse trees.

The semantic analyzer discards the inappropriate segment meanings or parse trees by doing the consistency check. If the action's owner and the parameters don't match the subject and targets filled in the HT, then

- the segment meaning filled in the **Subject Name, Target1 Name, Target2 Name** or **Action Name** is incorrect, or
- the parse tree is incorrect.

The semantic analyzer has to choose another segment meaning or parse tree to do the semantic analysis again until the right meaning or parse tree is selected, or the semantic analysis failed in all the possible choices.

## **6.5 The Verification of the Software**

To verify if the VTO system achieves the design objective, we input every rewritten policy listed in Appendix C to the VTO system and checked if the output FR format policies match the formal statements given in Section 5.1.6.5 in Chapter 5. The results generated by the system is exactly the same as the FR format policies we manually wrote in Section 5.1.6.5 in Chapter 5. The prototype is proven to be correct.

## Chapter 7

### Conclusion And Future Work

In this chapter the conclusions of the thesis are summarized and some suggestions for areas of further work are given.

#### 7.1 Conclusions

The aim of this thesis is to present two levels of policy specifications for both human and the software processing and introduce a methodology to construct a Restricted Natural Language (RNL) and the corresponding interface (RNLI) for policy capturing. The thesis first investigated the existing problems in policy capturing area, then the development issues for natural language interface design is examined. Based on these investigations, two levels of policy specifications are presented and a methodology of how to construct a RNLI is introduced. As the examples of how to use this methodology, a VTO system and a VCC system are illustrated.

Although policy-based systems are gaining more importance recently, the research on policy capturing stays in an early stage and no ease-of-use and ease-of-learning tools are developed to capture policies. Policy-based systems are expected to play a role not only in the system management but also in satisfying special needs of the end users. We expect that the need for an easy-to-use and easy-to-learn policy-capturing tool will increase in the future. In this context the RNL-based policy specification could play a role. The FR level specification, which is derived automatically from RNL, could be used for stepwise refinement and conflict analysis etc.

The RNL level policy specification represents policies using a Restricted Natural Language. The RNL is a subset of the natural language and is restricted in the lexicon, the grammar, as well as the semantics. It is used for the end users to input their policies into a policy-based system.

The RNL is restricted because both the lexicon and the grammar are selected subsets of the natural language; and the semantics describes the knowledge only in one problem domain. How good the “habitability” ([44] and [38]) of the RNL is depends on what words and syntax are chosen. On the other hand, a well-chosen lexicon and syntax also reduce the ambiguity problems of the RNL.

The FR (Formal Form) level policy specification express policies using a logic-based formal representation. Instead of giving end users the convenience to input their policies, the formal form of the policy specification focuses on the preciseness of the information in order to enable the automatic processing by software. In this thesis, the policies are stored in FR format in the policy repository, and is ready for the further automatic processes such as automatic policy refinement, static policy conflict analysis and policy enforcement etc.

The RNLI designed based on the RNL has two main objectives:

- (1) to be a user interface for the end users to use the RNL.
- (2) and to translate the RNL format policies into the FR format policies and store them into the policy repository.

A RNLI consists of a lexicon for the specific domain, a grammar for constructing policy statements, an object-oriented model for representing the knowledge expressed in the lexicon, (these three components combines a RNL), a formal form of the policy representation for the behavior of the specific domain, and a parsing technique to map the RNL into the formal form of the policy representation.

Inside the RNL parser, which is an essential part of a RNLI, we developed a two-phase lexical analysis to tolerate the spelling errors in the input sentences and use a Hidden Template (HT) to help the translation from the RNL to the FR.

The methodology of constructing the RNLI is demonstrated through the VTO system and the VCC system which we believe that this generalized and methodical approach could be used in.

## 7.2 Future Work

There are a number of areas in which this work can be expanded. In the area of the natural language interface, if the RNLI is combined with commercially available speech recognition engines to provide spoken input, it will minimize the training requirement of the end users and support hands-free or mobile usage. In the area of policy management, a better policy editor tool could be developed if we integrate the RNLI with a policy refinement tool and a conflict analyzer.

The well-known concept of “habitability” is examined in order to measure the productivity of our Restricted Natural Language. The habitable degree of a system is determined by the way we choose the lexicon and the grammar rules. However, we do not have the tools to quantitatively measure the habitable degree of a system. Furthermore, following what kind of rules can improve the habitability of a system still needs investigation.

There is also room for improving of our VTO system’s RNL Processor. In our lexical analyzer, the similarity measurement algorithm is only used in the word level. As the result, when errors exist in the input sentences, only those single word spelling errors can be detected and the right words can be suggested. If we could expand this similarity measurement algorithm to a segment level, the habitability of the language will be improved.

Finally, because of the limitation of the RNL’s lexicon, syntax and semantics, the information a RNL can express is limited. How to express as much information as possible and maintain the small size of the lexicon, syntax and semantics at the same time is always an issue we should examine. Although different RNLs have different lexicon, syntax and semantics, we believe that some basic parts are similar. For example, we believe that the core grammar of different RNLs should be similar. If we could find out this similar part



and provide a general solution to it, the challenges of constructing different RNLs would get simpler. How to find out this core part, certainly, needs more work.

## Bibliography

- [1] A. V. Aho and J. D. Ullman. The Theory of Parsing, Translation, and Compiling. Prentice-Hall, Inc., 1972.
- [2] A. V. Aho and J. D. Ullman. Principles of Compiler Design. Addison-Wesley Publishing Company, 1977.
- [3] J. Allen. Natural Language Understanding. Springer-Verlag, 1st edition, 1987.
- [4] D. Amyot. Use case maps quick tutorial version 1.0. <http://www.usecasemaps.org>, 1999.
- [5] D. Amyot and G. Mussbacher. On the extension of uml with use case maps concepts. In The 3rd International Conference on the Unified Modeling Language (<<UML2000>>), York, UK, October, 2000.
- [6] A. W. Appel. Modern Compiler Implementation in Java. Cambridge University Press, 1998.
- [7] B. Bruce. Case systems for natural language. Artificial Intelligence, vol. 6:pp. 327-360, 1975.
- [8] N. Chomsky. Three models for the description of language. IRE Transactions on Information Theory, vol. it-2(no. 3):pp. 113-124, 1956.
- [9] D. M. Cleal and N. O. Heaton. Knowledge-Based Systems: Implications for Human-Computer Interfaces. Ellis Horwood Limited, 1988.
- [10] R. Cole, J. Mariani, H. Uszkoreit, G. B. Varile, A. Zaenen, and A. Zampolli eds. Survey of the State of the Art in Human Language Technology. Cambridge University Press, 1998.
- [11] F. Cuppens and C. Saurel. Specifying a security policy: A case study. In Proc. of the Computer Security Foundations Workshop, pages pp. 123-134, Kerry, Ireland, 1996. 9th IEEE.
- [12] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In Proc. Policy 2001: Workshop on Policies for Distributed Systems and Networks, pages pp. 17-28, Bristol, UK, Jan. 2001. Springer-Verlag LNCS 1995.
- [13] R. Darimont and A. Lamsweerde. Formal refinement patterns for goal-driven requirements elaboration. In Proceedings 4th ACM Symposium on the Foundations of Software Engineering (FSE4), pages pp. 179-190, San Francisco, October 1996.

- [14] A. Dix, J. Finlay, G. Abowd, and R. Beale. HCI-User Centered Design. Prentice Hall Europe, 2nd edition, 1998.
- [15] C. Fillmore. The case of case. In P. Cole and J. Sadock (eds.), editors, Universals in Linguistic Theory. New York:Holt, Rinehart, and Winston, 1968.
- [16] Internet Engineering Task Force. <http://www.ietf.org>.
- [17] J. J. Granacki, Jr., and A. C. Parker. Phran-span: A natural language interface for system specifications. In Proceedings 24th ACM/IEEE Design Automation Conference, pages pp. 416-422, 1987.
- [18] C. Grossner, X. He, B. Kurusetty, G. Mahoney, and T. Radhakrishnan. Restricted natural language and an intermediate representation in policy controlled systems. Nortel Networks, Ottawa and Concordia University, 2001.
- [19] B. Grosz, D. Appelt, P. Martin, and F. Pereira. Team: An experiment in the design of transportable natural-language interfaces. Artificial Intelligence, vol. 32:pp. 173-243, 1987.
- [20] J. A. Hendler. Lecture notes of natural language interfaces, 1989.
- [21] S. E. Hudson. Cup parser generator for java. <http://www.cs.princeton.edu/apel/modern/java/cup/>. 1999.
- [22] Steven C. Johnson. Yacc: Yet another compiler compiler. In UNIX Programmer's Manual, volume 2, pages 353-387. Holt, Rinehart, and Winston, New York, NY, USA, 1979.
- [23] Venkatachalam Kanthimathinathan. Master's thesis: An enterprise policy specification tool. Concordia University, 2001.
- [24] M. Kay. The mind system. In R. Rustin, editor, Natural Language Processing. New York: Algorithmics Press, New York, 1973.
- [25] T. Koch, C. Krell, and B. Kramer. Policy definition language for automated management of distributed systems. In Proc. of Second IEEE International Workshop on System Management, June 1996.
- [26] R. Krovetz and W. B. Croft. Lexical ambiguity and information retrieval. ACM Transactions on Information Systems, vol. 10(no. 2):pp. 115-141, 1992.
- [27] A. Lamsweerde, R. Darimont, and P. Massonet. Goal-directed elaboration of requirements for a meeting scheduler: Problems and lessons learnt. In Proceedings RE'95 - 2nd IEEE Symposium on Requirements Engineering, pages pp. 194-203, York, March 1995.
- [28] P. Linz. An Introduction to Formal Languages and Automata. Jones and Bartlett Publishers, 2nd edition, 1997.
- [29] J. Lobo, R. Bhatia, and S. Naqvi. A policy description language. In Proc. of AAAI, Orlando, Florida, USA, July 1999.

- [30] E. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. IEEE Transactions on Software Engineering, vol. 25(no. 6):pp. 852-869, 1999.
- [31] B. Macias and S. G. Pulman. A method for controlling the production of specifications in natural language. The Computer Journal, vol. 38(no. 4):pp. 310-318, 1995.
- [32] Z. Manna and A. Pnueli. The Temporal Logic of Reactive and Concurrent Systems Specification. Springer-Verlag, 1992.
- [33] M. Masullo and S. Calo. Policy management: An architecture and approach. In IEEE First International Workshop On Systems Management, Los Angeles, California, April 1993.
- [34] J. Moffett. Policy hierarchies for distributed systems management. IEEE JSAC Special Issue on Network Management, vol. 11(no. 9), Dec. 1993.
- [35] J. Mylopoulos. A language facility for designing database-intensive applications. ACM Transactions on Database Systems, vol. 5:pp. 185-207, June 1980.
- [36] H. A. Napier, D. M. Lane, R. R. Batsell, and N. S. Guadango. Impact of a restricted natural language interface on ease of learning and productivity. Communications of the ACM, vol. 32(no. 10):pp. 1190-1197, 1989.
- [37] K. K. Obermeier. Natural Language Processing Technologies in Artificial Intelligence: The Science and Industry Perspective. Ellis Horwood Limited, Chichester, 1989.
- [38] W. C. Ogden. Handbook of Human-Computer Interaction, chapter 13, pages pp. 281-298. Elsevier Science Publishers B. V. (North-Holland), 1988.
- [39] M. Osborne. Estimation of stochastic attribute-value grammars using an informative sample. In Proceedings of COLING 2000, Saarbruecken, Germany, 2000.
- [40] J. Preece, Y. Rogers, H. Sharp and D. Benyon, S. Holland, and T. Carey. Human-Computer Interaction. Addison-Wesley Publishing Company, 1994.
- [41] T. Ryutov and C. Neuman. Representation and evaluation of security policies for distributed system services. In Proceedings of the DARPA Information Survivability Conference Exposition, Hilton Head, South Carolina, January 2000.
- [42] H. Tanaka. Current trends on parsing - a survey. <http://citeseer.nj.nec.com/tanaka93current.html>.
- [43] Masaru Tomita. Efficient Parsing for Natural Language: A Fast Algorithm For Practical Systems. Kluwer Academic Publishers, 1986.
- [44] W. C. Watt. Habitability. American Documentation, vol. 19(no. 3):pp. 338-351, July 1968.
- [45] R. Wies. Using a classification of management policies for policy specification and policy transformation. In Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management, Santa Barbara, California, USA, May 1995. Elsevier Science Publishers.

- [46] W. Winiwarter. Adaptive natural language interface to faq knowledge bases. Data & Knowledge Engineering, vol. 35:pp. 181-199, 2000.
- [47] W. A. Woods. Transition network grammars for natural language analysis. Communications of the ACM, vol. 13:pp. 591-606, 1970.
- [48] W. A. Woods. An experimental parsing system for transition network grammars. In R. Rustin, editor. Natural Language Processing. Algorithmics Press, New York, 1973.

## **Appendix A**

### **The regulations for the examinations of the master's theses**

**The following regulations are extracted from the Concordia University's Graduate Calendar 1999-2000 in page 511-512**

Upon receipt of the thesis from the Thesis Office, the graduate studies committee of the program in which the candidate is enrolled appoints an examining committee in consultation with the thesis supervisor. The examining committee consists of a minimum of three, and a maximum of five members. The candidate's supervisor is a member of the examining committee. In the programs where there is a thesis supervisory committee, any or all members may be named as members of the examining committee, subject to the policy of the program concerned. SIP students must have one external examiner on their committee. Students in the M.A.Sc. programs in Engineering must have an examiner from outside their department.

The defence normally shall be scheduled within a period no fewer than two, and no more than five, weeks from the submission of the thesis. The parties concerned may agree upon a postponement.

The examining committee, and the thesis defence examination, will be chaired by a person appointed by the program graduate studies committee. The chair will act as a neutral person. Each member of the examining committee must submit a written report on the thesis to the chair before the defence. Any member of the University is free to attend a master's oral thesis defence. The chair will give priority to questions from members of the examining committee. Only members of the faculty of the University may be recognized by

the chair. The chair adjourns the examination when the examining committee decides that further questioning is unnecessary. The deliberations of the examining committee are held in camera, that is to say, only appointed members of the examining committee are present. It is the responsibility of the chair to see that a report on the examination is prepared before the committee adjourns. The examiner's evaluation forms and the examination report must be signed and returned to the thesis officer.

Four decisions are open to the examining committee, voting to be based on a simple majority. The thesis can be accepted as submitted, accepted with minor modifications, accepted with major modifications, or rejected. Minor modifications are defined as corrections which can be made immediately to the satisfaction of the supervisor. Major modifications are defined as corrections requiring further research, or structural changes, or other substantive revision. When a thesis is accepted with major modifications, a precise description of the modifications must be included in the examining committee's report along with a date for their completion. It is then the responsibility of the candidate's supervisor to demonstrate to the examining committee that the required modifications have been made. The candidate's supervisor should inform the Thesis Officer in writing that the modifications have been approved.

If the examining committee is not prepared to reach a decision concerning the thesis at the time of the thesis defence, it is the responsibility of the chair to determine what additional information is required by the committee to reach a decision, to arrange to obtain this information for the committee, and to call another meeting of the committee as soon as the required information is available. It is also the chair's responsibility to inform the candidate that the decision is pending.

## **Appendix B**

### **Class Definition**

In Appendix B, each class's definition, attributes and methods defined in the VTO system are listed.



policy	Noun	Object name	Inheritance	Attributes & Methods	
p		captured from policy p			captured from
p8	member of the examining committee	ECm	Person	Rep ECm rep	p8
				Signature signature	p15
				Eform ECm_ eform	p15
				Thesis thesis	p8
				bool Submit(Rep ECm_ rep, Chair c)	p8
				Rep WriteReport()	p8
				Question AskQuestion()	p10
				bool Attend(Deliberation d)	p13
				bool Sign(Doc d)	p15
				bool FillEform(Eform ECm_ eform)	p15
				bool AgreePostpone(Postpone p)	p6
p8	the examining committee	EC	n/a	String name	p8
				ECmSet ecm_ set	p8
				ModiResult modi_ result	
				Erep erep	p14
				Chair chair	p7
				Integer count	p1
				Info info	
				bool DecideEndQuestion()	p12
				bool PrepareReport(Erep p)	p14
				ThesisStatus VoteDecide(ThesisStatus status)	p16
				bool AddToReport(Modification m)	p17
				bool AddToReport(Time modi_ comp_ date)	p17
				ModiResult ApproveModification(Modification m)	p17
p10	chair	Chair	ECm		
				Question GivePriority(Question q)	p10
				Time Adjourn(Defence d)	p12
				bool CheckErep(Erep erep)	p14
				bool Return(Doc d, Tofficer o)	p15
				Info DeterAddiRep(Info in)	p18
				bool ArrangeObtain(Info info)	p18
				bool CallMeeting()	p18
				bool Inform(Stud s, ThesisStatus ts)	p18

p1	graduate studies committee	GSC	F	Program pgm	p1
				ECm CnlAppoint(Person person)	p1
				Chair AppointChair(ECm ecm)	p7
p1	member of graduate studies committee (added by myself)	GSCm	Fm		
p3	external examiner	Extm	Person	U unit	p3
p4	examiner outside the department	Outdeptm	Um	Department department	p4
p2	thesis supervisory committee	SupC	n/a	SupCSet SupC_set	p2
				Ingeter count	p2
p2	member of thesis supervisory committee (added by myself)	SupCm	Sup		
p17	supervisor	Sup	ECm	Stud stud	p1
				bool Demo(Thesis th, EC e)	p17
				bool Inform(Tofficer tof, ModiResult r)	p17
p17	candidate	Stud	Pm	Thesis thesis	p17
				SupC supc	p17
				Sup sup	
				GSC gsc	
				EC ec	p14
				Thesis ModifyThesis(Thesis t, Modification mo)	p17
				Thesis Submit(Tofficer of, Thesis th)	p1
p17	thesis	Thesis	n/a	Time submit begin_time	p1
				Defence defence	p5
					p12
				Deliberation deliberation	p13

				ThesisStatus status	p16
				EC ec	p17.1
p1	thesis office	Toffice	n/a	TofficerSet officer_set	p1
				Time Schedule(Defence d)	p5
				Postpone RequestPostpone(ECm m.Thesis th)	p6
				Time Postpone(Defence d, Postpone postpone)	p6
p14	thesis officer	Tofficer	Um	Toffice belongto	p14
p1	person	Person	n/a	String name	p1
p9	any member of the university	Um	Person	U unit	p9
				bool Attend(Defence d)	p9
p9	the university	U	n/a	UmSet um_set	p9
				String university_name	p9
p1	member of the department	Dm	Um	Department department	p11
p1	department	Department	U	String department_name	p1
				FmSet fm_set	p11
p11	member of faculty of the university	Fm	Dm	F faculty	p11
p11	faculty of the university	F	Department	FmSet fm_set	p11
				String faculty_name	p11
p4	program	Program	Department	String program_name	p4
				PmSet pm_set	p4
p4	member of the program	Pm	Dm	Program program	p4
p15	document	Doc	n/a	String sender_name	p15
				String receiver_name	p15
				Signature signature	p15
p8	written report	Rep	Doc		p8

p14	examining report	Erep	Doc	SignatureSet signature_set	p14
				Modification modification	p17
				Time modi_comp_date	p17
p15	evaluation form	Eform	Doc		
	thesis defence	Defence	Thesis	Time defence begin time	
				Time adjourn begin time	
	thesis deliberation	Deliberation	Thesis	Time adjourn begin time	
				Time adjourn end time	
		ThesisStatus	n/a		
		ModiResult	n/a		
		Modification	n/a		
		Set	n/a		
		ECmSet	Set		
		SignatureSet	Set		
		UmSet	Set		
		SupCSet	Set		
		FmSet	Set		
		PmSet	Set		
		TofficerSet	Set		
		Time	n/a		
p18	message	Message	n/a		
p18	additional information	Info	Message		
p6	postponement	Postpone	n/a		
p10	question	Question	n/a		

## Appendix C

### A Complete List of Re-written Policies Using VTO System's RNL

P1: Upon receipt of the thesis from the Thesis Office, the graduate studies committee of the program in which the candidate is enrolled appoints an examining committee in consultation with the thesis supervisor. The examining committee consists of a minimum of three, and a maximum of five members. The candidate's supervisor is a member of the examining committee.

**Re-written policy:**

P1.1: upon receipt of the thesis from the thesis office, the graduate committee of the program in which the candidate is enrolled should appoint the candidate's supervisor to be a member of the examining committee.

P1.2: upon receipt of the thesis from the thesis office, the graduate committee of the program in which the candidate is enrolled should appoint a person to be a member of the examining committee in consultation with the thesis supervisor.

P2: In the programs where there is a thesis supervisory committee, any or all members may be named as members of the examining committee, subject to the policy of the program concerned.

**Re-written policy:**

P2: In the program where there is a thesis supervisory committee, the graduate committee of the program may appoint any or all members of the thesis su-

pervisory committee to be the member of the examining committee.

P3: SIP students must have one external examiner on their committee.

**Re-written policy:**

P3: The graduate studies committee must appoint an external examiner for SIP students.

P4: Students in the M.A.Sc. programs in Engineering must have one examiner from outside their department.

**Re-written policy:**

P4.1: The graduate studies committee must appoint an examiner outside the department if the student is in the M.A.Sc program in Engineering.

P4.2: The graduate studies committee must appoint an examiner outside the department for the student in the M.A.Sc program in Engineering.

P5: The defence normally shall be scheduled within a period no fewer than two, and no more than five, weeks from the submission of the thesis.

**Re-written policy:** Invalid.

P6: The parties concerned may agree upon a postponement. **Re-written policy:**

P6.1: The thesis office may request a postponement addressed to the members of the examining committee.

P6.2: The members of the examining committee may agree upon the postponement.

P6.3: If the members of the examining committee agree upon the postponement, the thesis office should postpone the thesis defence.

P7: The examining committee, and the thesis defence examination, will be chaired by a person appointed by the program graduate studies committee. The chair will act as a neutral person.

**Re-written policy:**

P7: The program graduate studies committee must appoint a chair.

P8: Each member of the examining committee must submit a written report on the thesis to the chair before the defence.

**Re-written policy:**

P8: Each member of the examining committee must submit a written report on the thesis to the chair before the defence begins.

P9: Any member of the University is free to attend a master's oral thesis defence.

**Re-written policy:** Same as above.

P10: The chair will give priority to questions from members of the examining committee.

**Re-written policy:** Same as above.

P11: Only members of the faculty of the University may be recognized by the chair.

**Re-written policy:** Invalid action. The policy is discarded.

P12: The chair adjourns the examination when the examining committee decides that further questioning is unnecessary.

**Re-written policy:** Same as above.

P13: The deliberations of the examining committee are held in camera, that is to say, only appointed members of the examining committee are present.

**Re-written policy:**

P13: Only appointed members of the examining committee are present in the deliberations of the examining committee.

P14: It is the responsibility of the chair to see that a report on the examination is prepared before the committee adjourns.

**Re-written policy:**

P14.1: The examining committee should prepare a report before the committee adjourns.

P14.2: The chair should check the examining committee's report.

P15: The examiner's evaluation forms and the examination report must be signed and returned to the thesis officer.

**Re-written policy:**

P15.1: Each examiner must sign the examiner's evaluation form.

P15.2: Each examiner must sign the examination report.

P15.3: The chair must return the examiner's evaluation forms to the thesis officer.

P15.4: The chair must return the examination report to the thesis officer.

P16: Four decisions are open to the examining committee, voting to be based on a simple majority. The thesis can be accepted as submitted, accepted with minor modifications, accepted with major modifications, or rejected. Minor modifications are defined as corrections which can be made immediately to the satisfaction of the supervisor. Major modifications are defined as corrections requiring further research, or structural changes, or other substantive revision.

**Re-written policy:**

P16: The examining committee vote to get the final decision.

P17: When a thesis is accepted with major modifications, a precise description of the modifications must be included in the examining committee's report along with a date for their completion. It is then the responsibility of the candidate's supervisor to demonstrate to the examining committee that the required modifications have been made. The candidate's supervisor should inform the Thesis Officer in writing that the modifications have been approved.



**Re-written policy:**

- P17.1: When a thesis is accepted with major modification, the examining committee must include a precise description of the modifications in their report.
- P17.2: When a thesis is accepted with major modification, the examining committee must also include a date of modification completion.
- P17.3: the candidate should modify the thesis.
- P17.4: When a thesis is accepted with major modification, the candidate's supervisor should demonstrate to the examining committee that the required modification have been made.
- P17.5: The examining committee should give an opinion about the modification.
- P17.61: The candidate's supervisor should inform the thesis officer in writing that the modification has been approved.
- P17.62: If the modifications have been approved, the candidate's supervisor should write to the thesis officer about the result.
- P17.63: If the modification has not been approved, the candidate should continue to modify the thesis.
- P18: If the examining committee is not prepared to reach a decision concerning the thesis at the time of the thesis defence, it is the responsibility of the chair to determine what additional information is required by the committee to reach a decision, to arrange to obtain this information for the committee, and to call another meeting of the committee as soon as the required information is available. It is also the chair's responsibility to inform the candidate that the decision is pending.

**Re-written policy:**

- P18.1: If the examining committee is not prepared to reach a decision concerning the thesis at the time of the thesis defence, the chair should determine what additional information is required by the committee to reach a decision.

P18.2: If the examining committee is not prepared to reach a decision concerning the thesis at the time of the thesis defence, the chair should arrange to obtain this information for the committee.

P18.3: As soon as the additional information is available, the chair shall call another meeting of the committee.

P18.4: The chair should inform the candidate that the decision is pending.