A Study on Implementing User Requirements Using Policies and Standards

– Applied on Network Management Systems –

2002

Tetsuo Otani

Abstract

As software supports a wider range of work, it has become more difficult to elicit requirements and design functions rapidly and precisely. This trend is noticeable in the design of systems that support business processes requiring a significant amount of expertise. In such domains, designers sometimes have to understand users' expertise to read requirements for the functions to be built. In addition, this kind of expertise differs from one industry to another and from one organization to another. The requirements may change even after the functions are implemented. Further studies are required for rapid and precise design of functions even though there are many contributions to software engineering.

This thesis presents new methods for improving the situation. The key concepts of this thesis are function design led by users, adaptive functions, and customization of existing software with minimal influence. The function design led by users means that users play a leading role in designing functions. Designers do not elicit requirement from users, but users draw up specifications of functions they need for the sake of reflecting users' idea. The concept of adaptive functions includes two aspects, adaptability to the change of requirements and of conditions. These two aspects eliminate or reduce work for change design or implementation of functions. In cases where functions have to be changed, reuse of existing software is important for rapid development. The influence should be minimal for the reuse of existing software. Minimal influence means low volume of work for modifying the existing software. It is more significant if the existing software conforms to a standard. The updated version of the existing software will be able to be used easily if the existing software is not changed.

We focus on the elicitation of requirement for and the design of functions in a network management system (NMS) in this thesis. This domain is rich in standards for design and implementations of functions. These standards will help rapid development of functions. In addition, these functions are designed based on much expertise in the domain so this is one of the domains in which designers have trouble in eliciting requirement and developing functions.

For the function design led by users, we have developed a technique in which users draw up use cases detailed by policies. Use case method is a major annotation for object-oriented analysis and design. It is represented as an ellipse with a simple instruction showing a provided service. This simplicity allows users to use this annotation. The details of a use case are drawn up with policies, which are rules of network management. We have offered several types of policies to represent various rules, and two types of formats to describe policies. We have applied this technique to a case study in which network operators draw up specifications of functions supporting scheduling of maintenance tasks.

For the adaptive functions, we have developed a mechanism for policy selection. This mechanism has adopted a concept of an immune network that can accept changes of policies and recognize changes of conditions in networks or services. The results of experiments in which the mechanism was applied to the scheduling tasks show that network operators can add,

change or remove policies without changing the core part of the mechanism for policy selection. Furthermore, the results are validated by the operators that work for real business.

A policy selected by the mechanism, calls a function provided in a system. If existing software has provided the function, a designer only connect the policy with the function. If not, the designer has to develop a new function for the policy. This development is supported by the concept of customization of exiting software with minimal influence. We offer two techniques for the customization, RevComponents and Partial Extension Package (PEP). RevComponents are software components that add, change, or remove some features of a function provided by an off-the-shelf component. They can also adapt an off-the-shelf component to a changed interface of another component, and distribute information to others. RevComponents can perform without change of source codes in the relevant off-the-shelf component. PEP supports addition of new features to object classes with minimal influence. It provides a scheme for setting new processing, a new state for invoking the new processing, and additional data needed by the new processing. These techniques have been validated through experiments in which the behaviors of existing software were changed using these techniques.

Through the proposed techniques, design of functions in network management systems becomes more rapid and more precise. These techniques would be suited to other domains that have characteristics similar to the network management. For example, an operation of Electric Power Systems would be the most promising domain. Each technique in this thesis can be used individually. We believe that our techniques contribute to realize the world in which an expert can design a function they need and share valuable software.

Key words

Use case, Policies, Artificial immune networks, Design patterns, Network management, Off-the-shelf components, International standards

Acknowledgement

I would like to thank all people who made this work possible by helping shape my research vision, sharing insights, discussing ideas, reviewing papers, and encouraging me.

I am deeply grateful to my advisor, Professor Yoshikazu Yamamoto, for his invaluable guidance, support and patience during my Ph. D. program. He was always open to discussions, and gave me the freedom and means to mature my ideas. Especially, he introduced me to an artificial immune network, which was a key concept in this thesis.

My gratitude also goes to Prof. Norihisa Doi, Prof. Ken-ichi Harada, and Prof. Masafumi Hagiwara for serving on my dissertation committee. They invested the time to give me feedbacks for earlier drafts of this dissertation through their careful readings.

I would like to express my sincere appreciation to Dr. Hiroshi Kaminosono and Yukio Inoue, who gave me a change to study in the area of network management. The theme of my dissertation comes from this study. I would also like to thank to Yoshiyuki Takeda and Dr. Yoshizumi Serizawa who allowed me to go to the graduate school for PhD degree while I work for CRIEPI.

It is not possible to enumerate everybody who gave me suggestions about the work in network management. In particular, I would like to appreciate to Shigeru Miyazaki and Tadashi Kobayashi, who have validated the results of my experiments in this thesis.

Dr. Jun-ichi Suzuki, Nozomu Matsui, Shogo Tsuji, Hiroaki Fukuda, Nobuyuki Matsushita, gave me ideas and contributed to mature my ideas.

My parents have given me unrelenting support for longer than I remember. I cannot thank to them enough for their kindness. My wife, Miki, has supported my work anytime anywhere. My daughters, Yuina and Ayane, have always encouraged me for the busy days. I love my family.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

In this thesis, we will be on the subject of function design. The term "function design" and "design of function" in this thesis mean to define structure and behavior of software that provides the function. A function in this thesis is information processing of software to produce a meaningful output or to carry out a meaningful action. The design of functions of a computer system that fulfill user requirements has not yet become a light task, although we see many contributions to software engineering. One of the reasons causing this problem comes from that computer systems have been larger and more complex. These shifts make it difficult for system developers to understand user requirements precisely and in detail.

This trend is noticeable in the design of systems that support business processes requiring a significant amount of expertise. This kind of expertise differs from one industry to another and from one organization to another. For example, communication networks are managed with expertise in communications techniques, network configurations and so forth. Electric power systems, as another example, are kept going by using knowledge of dynamos, transmission lines and the like. Functions provided by a system supporting these kinds of business processes are designed based on expertise relating to an industry and/or an organization.

People working for such organizations are experts in a specific domain, however few of them have taken part in a project in which a large-scale system is constructed. Therefore, they are not good at presenting their requirements for a system to be built. On the other hand, system developers do not usually have the individual expertise or know the contexts of the user business before they contact users. Moreover, this kind of expertise is difficult for system developers to learn. Accordingly, they usually take a long time until the design stage, and may misunderstand user requirements for the functions to be designed.

In such domains, products defined as standard and commercial software components are offered. These products provide functions for common use in a domain. The cost of building a system would be reduced and the period would be shortened, if such products were used. Indeed, some of these functions fulfill user's requirements without change of the software. However, other functions provided in such products are required to be modified to satisfy the requirements.

We acknowledge two gaps in the design of function, as illustrated in Figure 1. One is the knowledge gap between a user and a system designer. The other is the functional gap between a

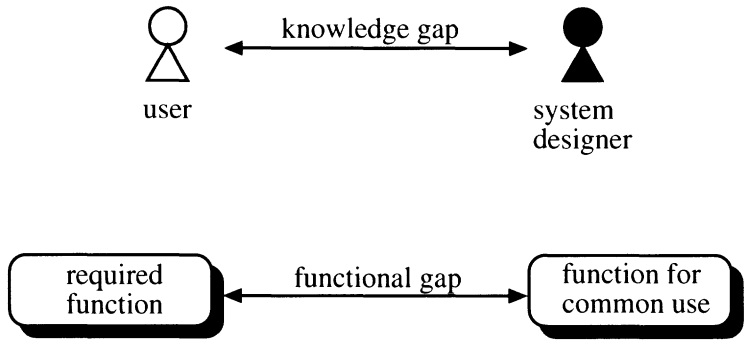


Figure 1: Two gaps in the function design

function required by a user and a function provided by a product for common use. The goal of our research is to bridge these gaps.

We propose methodology that improves the difficulty of function design caused by the problems as we mentioned above, and assists with fitting a function to a requirement in a short time. Figure 2 illustrates an overview of our proposal in this thesis. This proposal is composed of the three major parts, written in a bold type in Figure 2, described as follows.

1) A technique for requirement elicitation and description
2) A mechanism for selecting a subset of policies that meet a situation
3) Techniques for customizations of off-the-shelf components and the standard object classes.

1.1 Requirement elicitation and description

In our technique, a user takes the initiative in describing their requirements for functions to be designed. The purpose of this technique is to improve the problem of requirement elicitation. In other words, this bridges the knowledge gap between a user and a system designer. This technique does not replace traditional techniques in RE, but complements them. We refer to this technique as Function Design Led by Users (FDLU).

In FDLU, a user draw up a use case diagram [1, 2] that shows the functions to be designed. In the next place, a user details a use case using policies. A policy in this thesis declares a pair of a condition and an operation, or a relation between policies that are the first type. A condition and an operation are described at an actual task level. A condition stands for a state that invokes the operation described in the same policy. An operation in this thesis means a function provided

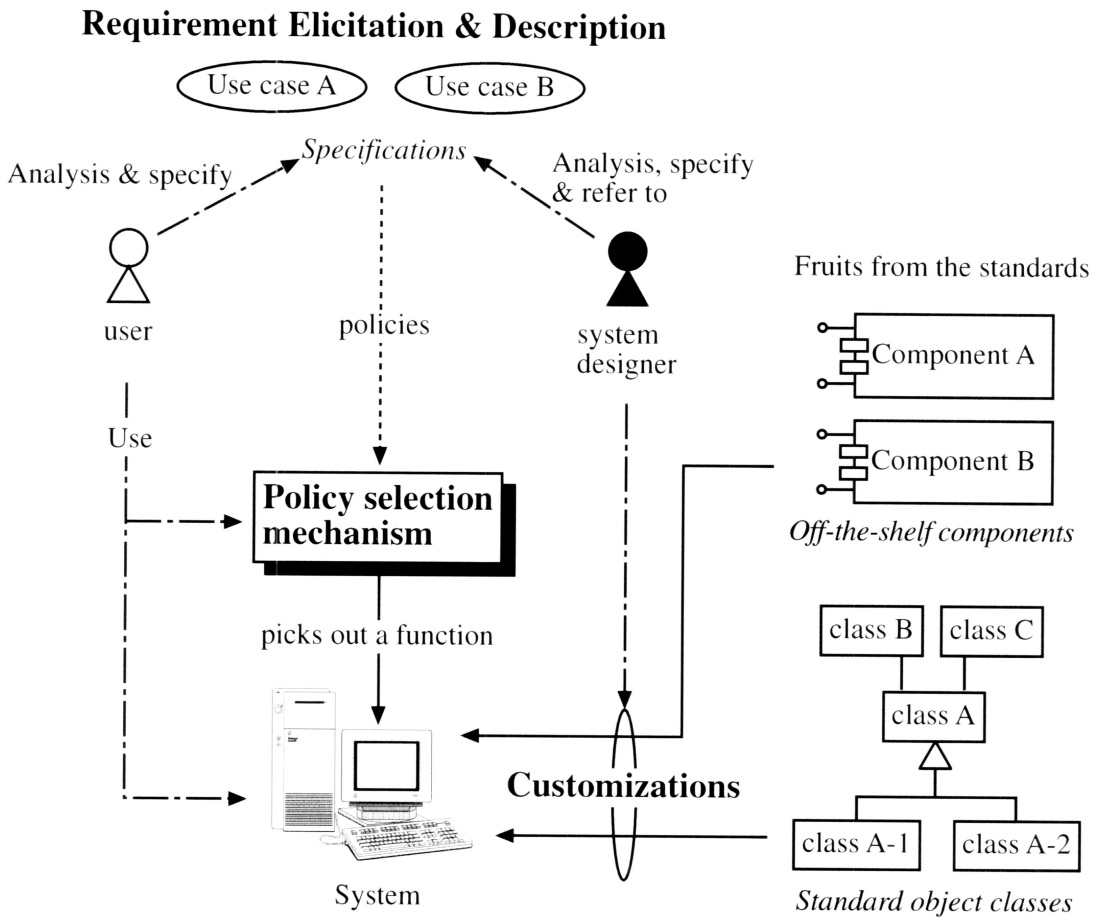


Figure 2: Overview of our proposal

by a system, a task carried out by a human user or a combination of them. FDLU provides two types of formats in which policies are declared, and defines rules how a policy is attached to a use case.

1.2 Policy selection

A policy selection mechanism picks out a subset of policies appropriate to the conditions in the networks or services to be managed [3]. The system including this mechanism performs based on the selected policies.

This mechanism adopts a concept of immune networks, in order to make an NMS adaptive to a change of user requirements and the situations in the network or services. In addition, the concept of immune networks contributes to shortening the processing time enough to be feasible to an actual business.

This mechanism eliminates or simplifies drawing up a sequence diagram related to a use case in the stage of requirement elicitation and description. It enables the system to which this method is applied to decide its behavior in each case. In this sense, this method has a good effect on the stage of requirement and analysis.

1.3 Customizations

We set out two techniques for the customization of functions. The first technique provides a method at the level of software components. The second one does a method at the object classes. In both cases, we make full use of products that are defined as a standard or are sold in commercial. This is for short period and low cost of building a system.

System designers make software programs providing a function that performs as defined in a policy. They can set an off-the-shelf component into a system if the component has already provided the function. An off-the-shelf component is a software part that can be used as it is. If not, they would develop a new program. In this case, the designers may change an existing off-the-shelf component, or create a new program from scratch using the standard classes. The first technique for the customizations allows designers to use an off-the-shelf component without change, and add new features. This technique offers four types of components for customization. We have named these components "RevComponent" [4].

In the case where designers create a new program from scratch using the standard classes, they can make use of the second technique that we offer. This technique sets a package in which classes can have connections to the standard ones with minimal influence. We refer to this technique as the Partial Extension Package (PEP) [5].

These two techniques enable the use of products from the standards if they are updated. They have good effects on the design stage, because they provide simple way to customize a function with minimal side effects on existing products. They bridge the gap between a function provided in a product for common use and an operation declared in a policy.

1.4 Usage of the proposed techniques

Figure 3 shows how to use the proposed techniques on the stages of a development process. This figure uses Rational Unified Process (RUP) [6] as the development process. RUP is currently one of the major processes based on the concept of object-oriented technology. In our methodology, a user takes part in the stages of requirement definition and analysis. He/she specifies functions using use cases defiled by policies. A system designer refers to this use cases and design functions. The designer may picks up a product for common use such as a standard object class or an off-the-shelf component, if a function specified by the user is fulfilled by it. If not, the designer customizes a function provided by a product for common use. Our methodology does not cover implementation or test of an object class. Issues about non-functional requirements, e.g. performance, are not covered in this thesis, either.

These techniques bridge the gaps mentioned above. Figure 4 shows conceptually bridges of the gaps and support of the policy selection. Use cases detailed by policies fills the knowledge gap between a user and a system designer. This technique assists users to get closer to the function design without additional knowledge for them. The policy selection enables policies to be used to draw up requirements for functions. It determines the sequence of processing when a system using this selection works. This policy selection makes a function to be adaptive, so a function to be design may be simple or the need to modify a function may be omitted. Two techniques for customization bridge the functional gap. They provide simple ways to modify functions as well as keeping products for common use reusable.

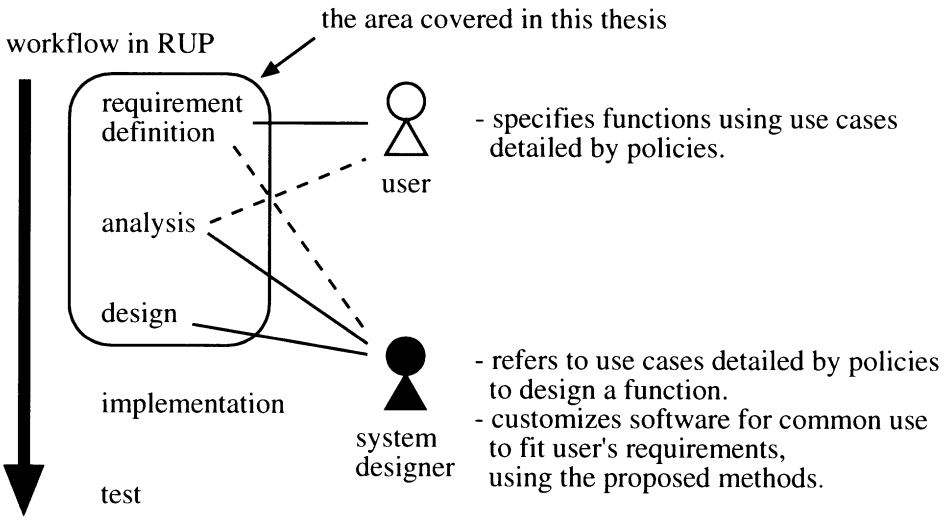


Figure 3: Usage of the proposed techniques on stages of a development process

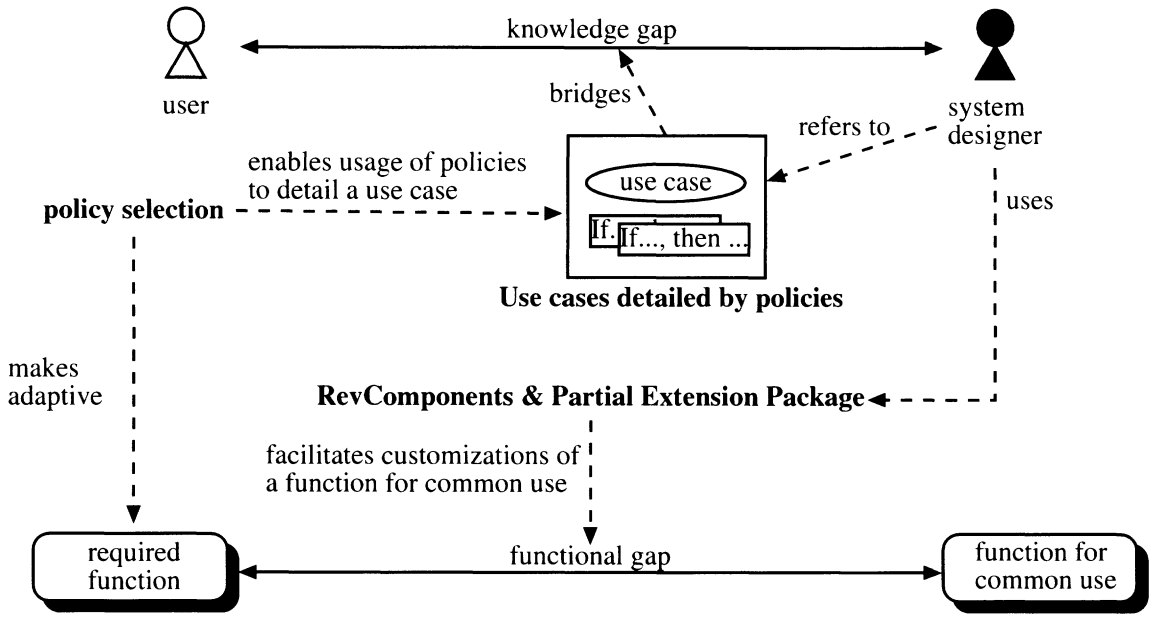


Figure 4: Bridges of the gaps using the proposed techniques

We will focus on the design of functions for network management since domains would be too various to cover. The reason why we have selected network management is that rich standards for this domain have already developed as compared with other industries. This enables us to make use of standard object classes and off-the-shelf components for system development.

The organization of this thesis is as follows. Chapter 2 shows an overview of network management and supporting systems, characteristics of function design, and traditional approaches to design functions. In Chapter 3, we will describe a method that enables users to describe user requirements. This method makes use of Use Case Diagrams detailed by policies, which are rules and/or directions to perform business processes in network management. Chapter 4 reveals a mechanism that picks out a subset of policies from a set, and decides the behavior of a system. This mechanism is adaptive to change of policies using artificial immune networks. Chapter 5 and Chapter 6 address techniques for the design of functions based on off-the-shelf software components and/or object classes set as international standards. These techniques have no or little impact on the object classes or the off-the-shelf components. This feature allows consecutive usage of the classes and components to consist with design of

functions specific to individual users. Future work is described in Chapter 8 and we will conclude this thesis in Chapter 7.

Chapter 2

Network management systems

The aim of this chapter is to clarify what network management is and to address the motivations that drive us to set up a new framework for function design. First, we give an overview of business processes and supporting systems in network management. Hereafter, we refer to these supporting systems as an NMS (Network Management System). Second, we address the characteristics of design process for functions in an NMS. We also review traditional approaches for the design of this kind of functions, and outline their problems.

2.1 Overview

Figure 5 gives a bird's eye view of the entire system that consists of several components and the staffs in network management. Telecommunication networks are composed of lines such as optical fibers or cupper lines, and nodes such as routers or transmitters. These provide end users with various kinds of communication services. This figure shows a Web server and its client as an example of a service.

Network management is a set of tasks whose aims are accurate and quick fulfillment, assurance and billing of services provided to users [7]. Network operators are responsible for achieving these tasks. They monitor and control networks, and issue instructions to the maintenance staffs in order to perform their duties. Networks are, however, usually too large and complex for the operators to monitor and control them without an NMS. An NMS collects information from network nodes and controls them by commands from network operators. It also stores the data for the management, for example, network configuration data or service lists.

In the following subsections, we will describe the details of management areas, business processes, network operators and an NMS.

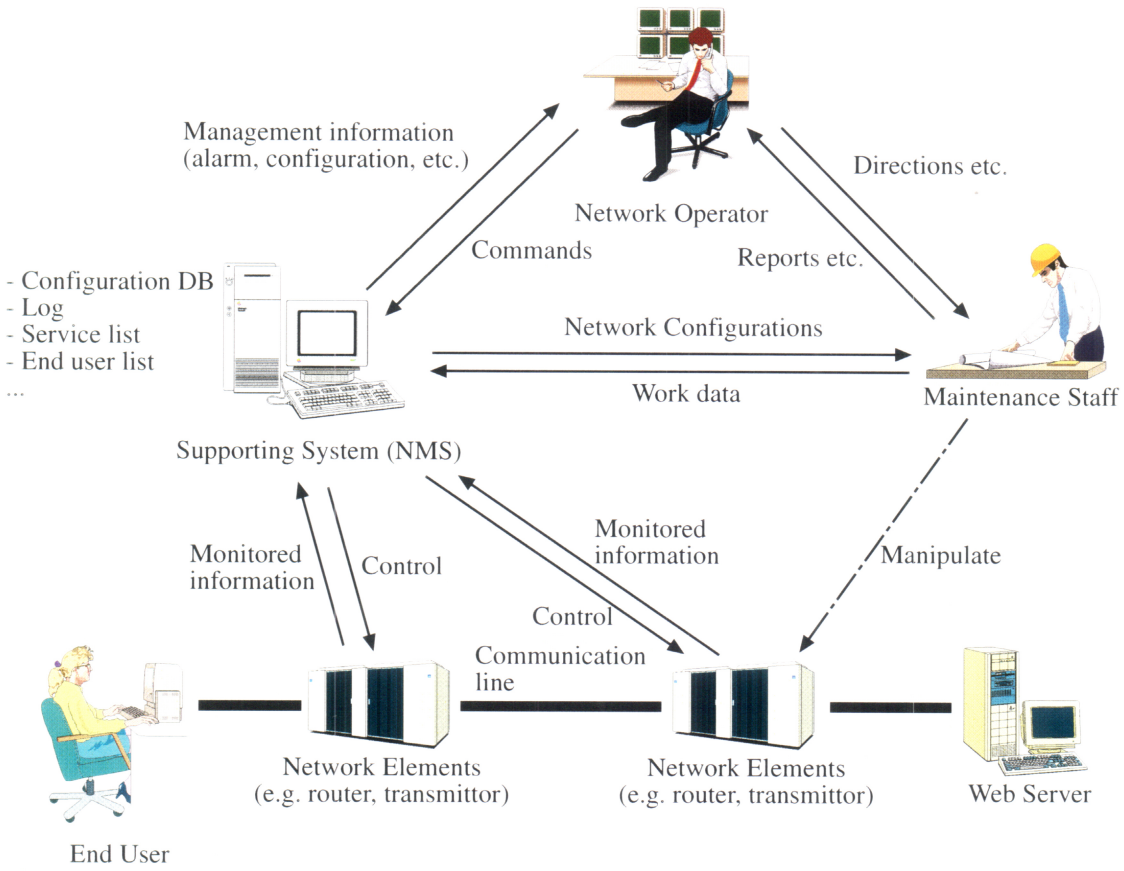


Figure 5: Staff, NMS, and telecommunication networks

2.1.1 Management areas

Tasks in network management are categorized into the following five areas [8]:

1) Fault management
2) Accounting management
3) Configuration management
4) Performance management
5) Security management

Fault management is to detect, isolate and correct a fault in networks. Faults cause networks to fail to meet their operational objectives and they may be persistent or transient.

Faults manifest themselves as particular events (e.g. errors) in the operation of a network node or line. Error detection provides a capability to recognize faults. Fault management includes the following functions.

a) Maintain and examine error logs.
b) Accept and act upon error detection notifications.
c) Trace and identify faults.
d) Carry out diagnostic test sequences.
e) Correct faults.

The accounting management enables charges to be established for the use of resources in networks, and for costs to be identified for the use of those resources. The accounting management includes the following functions.

a) Inform users of costs incurred or resources consumed.
b) Enable accounting limits to be set and tariff schedules to be associated with the use of resources.
c) Enable costs to be combined where multiple resources are invoked to achieve a given communication objective.

Configuration management includes identifying, exercising control over, collecting data from and providing data to networks. They are for initializing, starting, providing for the continuous operation of, and terminating services. Configuration management includes the following functions.

a) Set the parameters that control the routine operation of networks.
b) Associate names with elements to be managed
c) Initialize and close down elements to be managed.
d) Collect information on demand about the current condition of networks.
e) Obtain announcements of significant changes in the condition of networks.
f) Change the configuration of networks.

In performance management, the behavior of resources in networks and the effectiveness of communication activities are evaluated. This includes the following functions.

a) Gather statistical information.
b) Maintain and examine logs of system state histories.
c) Determine system performance under natural and artificial conditions.
d) Alter system modes of operation for conducting performance management activities.

The purpose of security management is to support the application of security policies by means of the following functions.

a) Create, delete and control security services and mechanisms.
b) Distribute security-relevant information.
c) Report security-relevant events.

2.1.2 Layer model

A layered model has been proposed in order to categorize business processes relevant to the management areas [9]. This architecture is composed of five layers, usually arranged in a triangle or pyramid as illustrated in Figure 6. Each layer interacts with neighboring layers in order to exchange information.

Tasks in the business management layer have responsibility for the total enterprise. They normally set goals rather than to achieve a goal, but can become the focal point for action in cases where executive action is called for. This layer is a part of the overall management of the enterprise and interacts frequently with other systems. The following tasks are examples in this layer.

1) Decision-making for the optimal investment and use of new telecommunications resources
2) Management of the budget related to OA&M (Operations, Administration and Maintenance)
3) Supply and demand of OA&M-related manpower
4) Aggregation of data about the total enterprise

The service management is concerned with, and responsible for, the contractual aspects of services that are being provided to customers or available to potential new customers. Some of

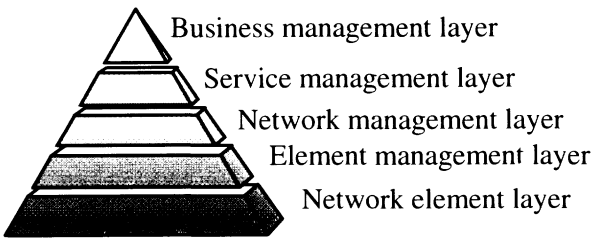


Figure 6: Management layer model

the main tasks of this layer are service order handling, complaint handling and invoicing. This layer has the following four principal roles.

1) Customer facing and interfacing with other operational organizations
2) Interaction with service providers
3) Maintenance of statistical data (e.g. Quality of Service)
4) Interaction between services

The network management layer has the following five principal roles.

1) The control and coordination of the network view of all network elements within its scope or domain
2) The provision, cessation or modification of network capabilities for the support of service to customers
3) The maintenance of network capabilities
4) The maintenance of statistical, historical and other data about the network
5) Interaction with the service management layer on performance, usage, availability etc. independent of technology

Thus, the network management layer provides the functionality to manage a network by coordinating activity across the network. Then it supports the "network" demands made by the service management layer. Fulfillment of tasks in this layer needs knowledge about what resources are available in the network, how these are interrelated and geographically allocated, and how the resources can be controlled. It requires an overview of the network. Furthermore, this layer is responsible for the technical performance of the actual network. In addition, it controls the capabilities and capacity of the network in order to give the appropriate accessibility and quality of service.

Tasks in the element management layer are performed to manage each network element such as a router. Information that this layer presents to the network management layer should be vendor-independent. Tasks in this layer have the following principal roles.

1) Control and coordination of a subset of network elements
2) Maintenance of statistical, log and other data about elements within its scope of control

2.1.3 Business processes

Tasks in the service and network management layers draw the attention of network operators. These tasks have a tendency to be specific to each organization while they contain functions that

can be shared between organizations. This is because they provide unique services to their customers using the same communication technologies. In addition, the aims of their tasks are similar, that is, they supply high quality service at low costs. On the other hand, tasks in the element management layer hinge mainly on the kind of and/or technology used in network elements. Tasks in the business management layer are influenced by factors other than ones of network management.

A business process model is set as a standard because tasks in the service and network management layer contain functions that can be shared between different organizations, and they have to interconnect their networks. Figure 7 shows a framework of business processes in

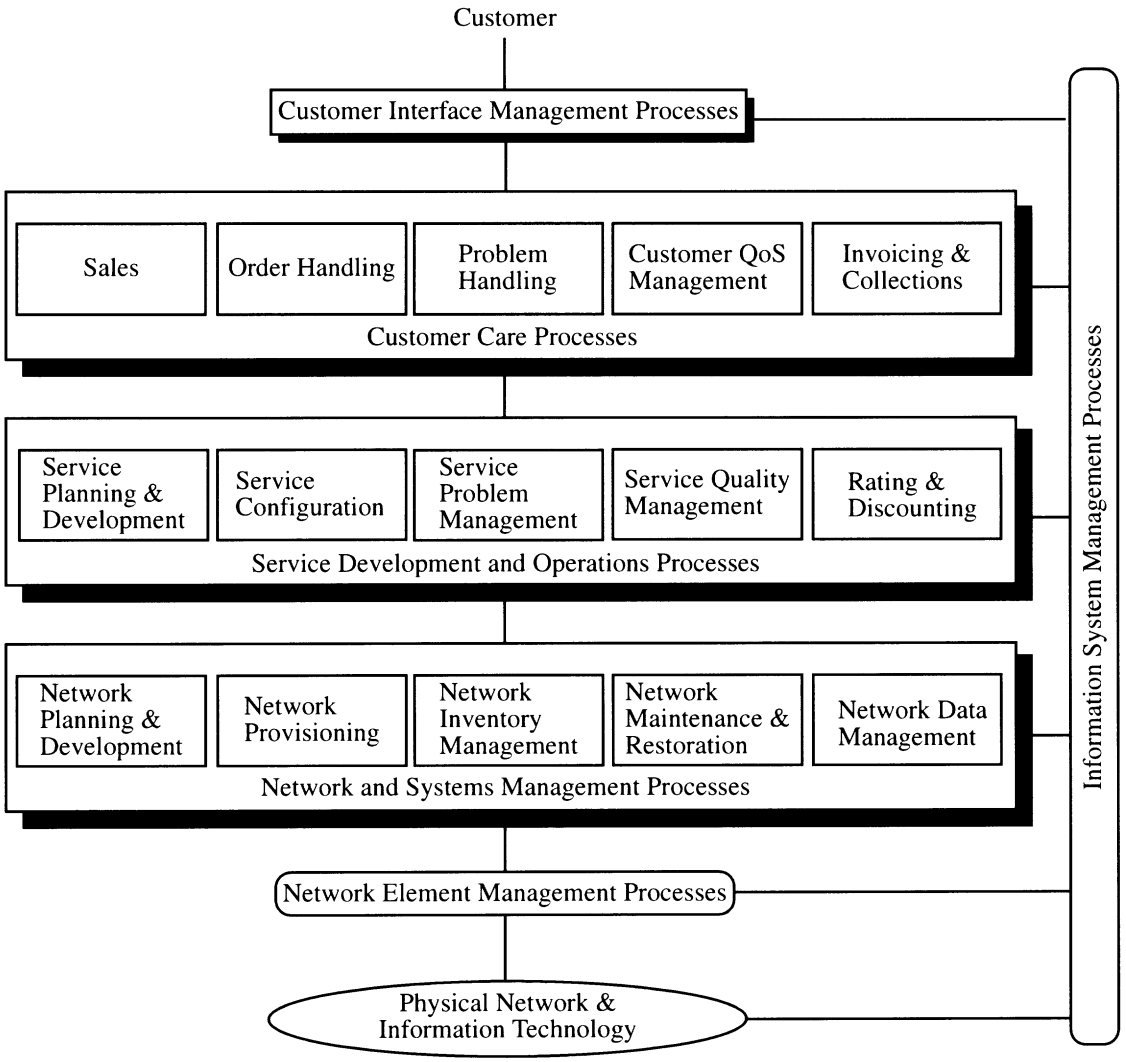


Figure 7: Telecom Operations Map, business process framework [7]

these two layers, which is defined by TeleManagement Forum (TMF) [7]. There are ten processes in the service management layer, and five processes in the network management layer. The service management layer is divided into two parts: "Customer Care Processes," and "Service Development and Operations Processes." In the simplest sense, the division reflects differences between processes triggered by individual customer needs and those applied to a group of customers subscribed to a single service or service family. These processes are service dependent and technology independent. On the other hand, "Network and Systems Management Processes" belong to Network Management Layer. They are technology dependent and vender independent. This framework is used for understanding the relationships among individual process flows.

Each process interacts with others. For example, Figure 8 shows data flows between processes for service quality management [10]. In some cases, a process gets some management information from another by sending a request. In other cases, a process gives notice to another periodically or due to an event. As other example, we can see other data flows for peer-to-peer service configuration [11], and order handling [12].

A business process is composed of several sub processes. In the case of the service quality management, there are eleven sub processes as illustrated in Figure 9. Functions in each sub process are performed by network operators, an NMS, or their corporations. Therefore, management data is passed via interfaces between human and machine, between software components, or between human beings. We can see the breaking down of other processes.

Many companies, including service providers and vendors, have complied with these business processes. They can enjoy the benefits of standard software based on these business processes if they accept the model. Network operators do not have any problem adopting the standard processes that are not used for their original services. They, however, need to develop some functions in business processes that are used for their original services. They need the original services in order to make themselves competitive in the telecom industry. Functions or tasks in the network management layer can be shared more easily than in the service management layer. This is because the service management layer has strong connections to their original services.

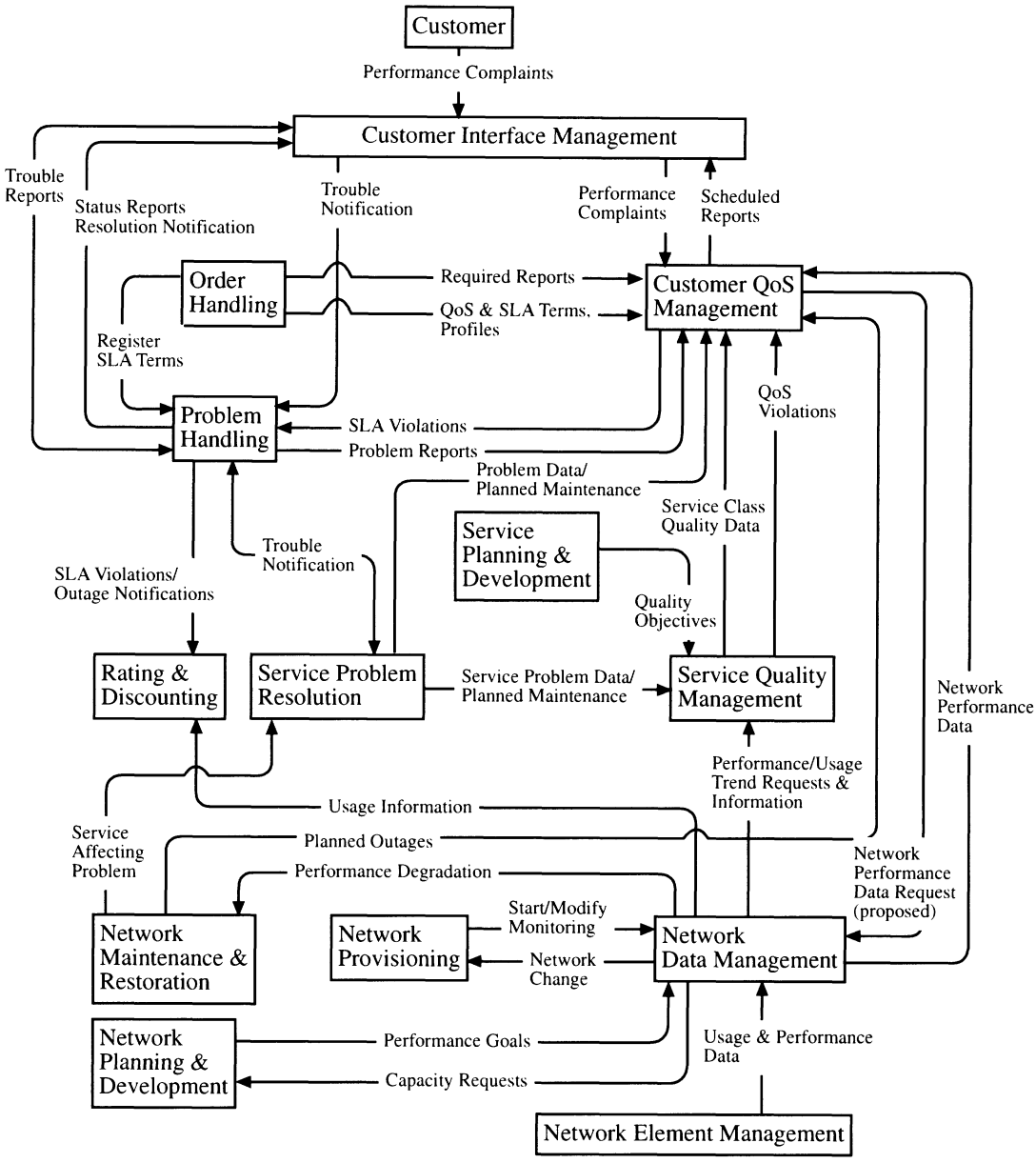


Figure 8: Performance management flows [10]

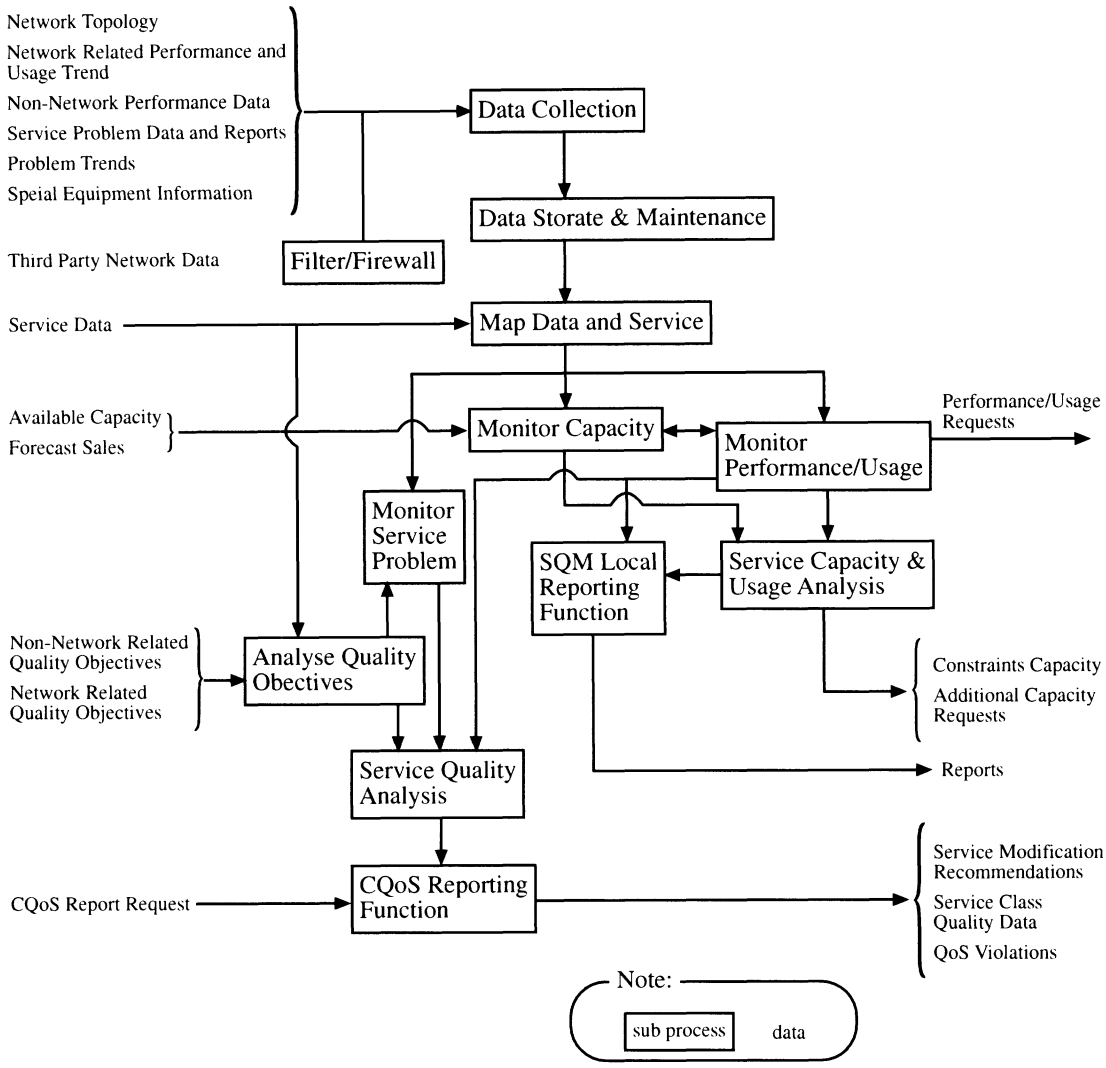


Figure 9: Sub processes in Service Quality Management [10]

2.1.4 Network operators

Network operators are usually experts in the management of networks and services that are own and provided by the organizations for which they work. They are familiar with how to read management information about their services and networks, and how to solve a problem occurring in the services and networks. Their knowledge and skills are gained from experience for a long period.

Network operators fill the jobs in one or several processes described in the previous subsection, through their knowledge and skills. They monitor the status of the service and networks. They may control elements of the networks or services if they find some malfunctions. They also have contact with customers for provisioning of and/or receiving complaints about the provided services. They direct staff in maintenance or sales sections to maintain service quality.

Network operators are also users of an NMS that monitor and control networks and services. They, however, are rarely experts in computer systems or software. Therefore, they do not know how to design functions in an NMS, or even how to represent their requirements to the systems. Nevertheless, they sometimes take part in a project in which the next generation support system is designed and developed. Such a project calls for their expertise in network management to design functions. This is one of the reasons why it takes a long time to construct an NMS.

2.1.5 NMS

An NMS executes functions supporting a part or whole of a business process or a sub process. It automatically processes management information based on strategies set by network operators, in the cases where a function supports the whole of a process or a sub process. It shows information to and receive inputs from network operators, in the cases where a function supports a part of a process or sub process. In both of these cases, an NMS interacts with network operators because it is required to provide human-machine interfaces.

Monitoring is the most basic function in an NMS. What kinds of data are collected depends on the service provided and/or network technologies. An NMS collects data about delays, delay fluctuations, loss rates and others of packets in IP networks. For telephone networks, an NMS gathers data about call loss rates and/or average duration of calls. Monitoring is started by a timer or request by network operators. The collected data is processed and used for representation to network operators or control of network elements.

An NMS may control network elements if these elements set parameters or start some activity due to messages from a remote machine. For example, an NMS may control the bandwidth of a logical path in ATM (Asynchronous Transfer Mode) networks. It may set routing tables in X.25 packet networks. Control is triggered by a result of analysis of monitored data, or

a request of network operators.

For the monitoring and control, an NMS has connections to network elements in order to gather and/or distribute data. Therefore, an NMS is usually distributed systems. There are several kinds of protocols between a network element and an NMS. On/off signals are the simplest way of communication between an NMS and network element. Proprietary protocols, such as CAPNET [13] and TL-1 [14], have been developed by several companies. In recent years, telecom companies have carried on the deployment of standard protocols such as SNMP (Simple Network Management Protocol) [15] or CORBA (Common Object Request Broker Architecture) [16].

An NMS stores a large amount of data as well as monitoring and controls. This data contains network configurations, service lists, customer lists, and so forth. In addition, an NMS saves monitored data such as error logs or historical data about traffic. These kinds of data are used for the analysis of performance, provisioning of new services, and/or accounting.

An NMS has recently been constructed based on several machines in order to disperse load caused by manipulation of the data. The machines may be located in a LAN, or on different sites. The protocols between these machines are the same as those mentioned above.

2.2 Characteristics of functions in an NMS

In this section, we address the characteristics of functions provided in an NMS. We can summarize the major features of these functions as follows.

1) Some functions can be shared between different network operators while others are specific to each operator.
2) Functions must be adaptive to conditions in networks or services.
3) Functions require expertise in order to be adaptive.
4) It is difficult to describe a sequence of a processing.
5) Requirements for a function may not appear at the time the NMS is built.

We will detail each characteristic in the following subsections.

2.2.1 Shared and original functions

Most functions that can be shared between different network operators are to provide primitive monitoring or controls. The primitive monitoring is to gather row data, such as delay time of packets, from network elements. The primitive controls are to set parameters or invoke an

activity on network elements. For example, routing parameters or bandwidth is set by this kind of control. Most of the primitive monitoring and controls relate to the network management layer or the lower layer. These functions are usually designed based on a certain communication technology such as ATM or IP. In addition, network operators need to change or replace these functions for original services.

Functions are inclined to be more distinctive as the management layer becomes higher. Most functions in the service management layer directly support one or several original services. Even functions in the network management layer may have unique feature for an original service. A service provider tries to be attractive to customers in its original services so that not all functions can be shared.

No one can decide which function can be shared or not. The decision is dependent on what kind of data, analysis and/or control is needed in the provided services. In addition, it is influenced by design policy. More shared functions can be used if network operators manually deal with processing that are not supported by the shared functions. More unique functions must be designed as an NMS provide more functions supporting original services.

2.2.2 Adaptation to various states

Each function in an NMS must be adaptive to a variety of conditions that occur in networks or services. One of the reasons why the conditions diverge is the number of customers. Using functions in an NMS, a network operator deals with many customers per service. He/she must respond carefully to customers' demands and/or complaints although the demands and complaints may be different from each other. This is because this response affects the credit of the service. The supporting functions should take account of the preference of each customer.

Various kinds of services also require functions in an NMS to be adaptive to a variety of states. Network operators may deal with different schemes for discount or compensation. In such case, functions in an NMS have to behave in light of the scheme to be applied. Network operators may have to ensure continued services that are provided to crucial business. Financial compensation for a halt of the services may have no effect in this case. Therefore, functions for the services may have to set and maintain redundant configurations of networks.

Functions in an NMS monitor and control a variety of network elements such as routers or transmitters. Monitored data is different due to the kind of technology or vendor. Control menus also differ from each other. Functions specific to an individual element do not need to take account of these differences. Functions relevant to several kinds of elements, however, may have to recognize the differences. If network configurations are changed dynamically using MPLS [17] or DiffServ [18], the functions may have to be more adaptive than ones for statically configured networks.

2.2.3 Expertise contained in functions

Functions in an NMS can be adaptive due to expertise that well-informed network operators have. Their expertise stems from carrying on actual operations for a long time. The operators have encountered states of affairs that had not been able to be predicted. They dealt with such states based on their skills and imaginations. Functions become more adaptive as they make better use of the expertise.

If functions are not designed based on expertise, they may have a red tape mind. If so, network operators would not make use of such functions. Functions that are not adaptive may not put up data that operators need. They may not control network elements to fulfill an operator's demands. Network operators would not rely on the functions that perform inappropriately even at low rates. Expertise is important for functions in this aspect.

Others, such as system designers or green operators do not have such expertise. Of course, the first reason is that they do not have rich experience in network operations. In addition, even well informed network operators cannot put down precisely on paper the expertise they have. This problem is caused by the complicated contexts in which the expertise was gained. Even if expertise is described precisely, others may not understand it. They do not have the rich experience to call upon in order to imagine the contexts in which the described expertise was gained.

Knowledge gained in a different environment cannot always be diverted because the contexts are different. However, it may help to understand or imagine the expertise specific to a network operator.

2.2.4 Difficulty of making a sequence of processing

The other feature in the design of functions in an NMS is the difficulty in defining a sequence of operations. Network operators can only draw a sequence of operations in a large sense. Such sequence is not elaborate enough to design functions. Operators must consider a large number of conditions used to determine which function is executed, if they detail the sequence. These conditions include one that rarely appears in the daily work of network operations. They have to concern themselves with a combination of the conditions.

In addition, a sequence of operations continually changes, even though users can specify the sequence. Operators have to address new customers, services or network elements. They have impacts on an order of operations and/or processing in an operation. Therefore, it may not be effective for the function design to detail a sequence of operations.

2.2.5 Appearance of requirements after NMS building

Network operators are users of an NMS providing functions that are designed based on their expertise. Therefore, they own the requirements to the functions to be designed. However, these requirements may change, or new requirements may appear after the NMS is built.

The first cause comes from new services. A service provider continues to develop new services in order to acquire new customers. Furthermore, services are being introduced more rapidly and with a shorter life span [19]. Every service is supported by functions in an NMS, and new services usually have some features different from existing ones. Therefore, functions must be changed to fulfill new requirements for new services.

New customers and technologies also result in new requirements for functions in an NMS. In particular, it is difficult for network operators to foresee what new customers will demand. Nevertheless, a service provider may not acquire new customers with their demands that new or modified functions will support, if response of the provider is not quick. Therefore, an NMS is required to provide new or changed functions quickly.

Network operator expertise is another type of cause that results in changing requirements for functions. As mentioned above, it is not easy to describe expertise precisely. Therefore, requirements to functions are incomplete at the stage of design. In many cases, network operators note that a function does not perform exactly as intended. Requirements to the function are fixed after it starts to be used.

The features we mentioned indicate that functions in an NMS should be changeable with little labor.

2.3 Design of functions

This section mentions the design of functions in an NMS. First, it surveys adopted approaches so far. This survey focuses on the elicitation of user requirements and the implementation of functions, which are important for NMSs. It describes standard techniques and off-the-shelf software components that can be used to construct an NMS. This sharing of design is a trend of the times for cost reductions and short periods of NMS constructions.

Through these surveys, we make it clear that there are three shortcomings of function design as follows.

1) Network operators cannot easily tell what functions they need. On the other hand, it is also difficult for system designers to understand the operators' requests.
2) We cannot make functions easily adaptive in an environment in which user requirements are changeable.

3) Few techniques are put forward to design individual functions based on shared software.

2.3.1 Elicitation of user requirements

Figure 10 shows the traditional exchange of information for function design. Network operators usually tell system designers of their images of functions to be made in a natural language. The system designers create formal specifications based on the image. If they have some questions about the image, they ask the network operators in the natural language, and clear up the questions. They suppose designed functions in the natural languages when formal specifications take shape. The network operators check and review the designed functions. They demand a change of the design if the design does not satisfy their requirements. Based on the requested change, the system designers modify the specification described formally. This cycle is continued until the network operators accept the design.

In these tasks, the system designers play the role of a translator of the natural language into the formal specifications, and vice versa. This is caused by the fact that few network operators are familiar with formal specifications. This type of interaction, however, raises a problem for the design of functions. In many cases, system designers are not familiar with business processes as the context of functions to be designed when they start the design. In addition, it is difficult for them to learn the processes and/or expertise, as we mentioned in Section 2.2.3. System designers, therefore, take a lot of work to elicit precise requirements for functions.

This problem is recognized in the Requirement Engineering (RE). Many tools and techniques have already been proposed to make a well-defined specification [20]. Moreover, it is recognized that RE contributes to success in a software project [21]. Getting requirements, however, is still a difficult part of a software project. It was reported that RE was deficient in

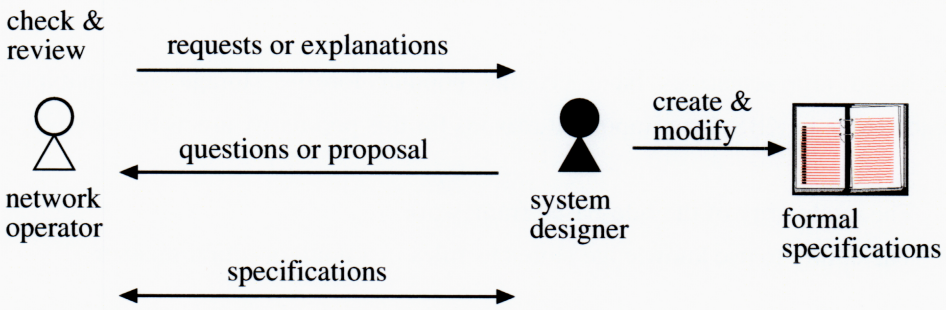


Figure 10: Interactions between an operator and system designer in traditional design

more than 75% of all enterprises [22]. One of the reasons why RE is insufficient in practice is the knowledge gap between users and system designers, as well as in the case of NMSs.

2.3.2 Implementations of functions

Several techniques make software adaptive to change. Typical cases are software agents and knowledge-based systems (KBS). We deal with these two techniques separately in this thesis, although these two techniques are not mutually exclusive.

With regard to agents, the key concepts are that agents can act autonomously to some degree, and they are part of a community in which mutual influence occurs [23]. Software agents have already been applied to a number of fields [24]. We can see examples of NMSs that adopt software agents [25, 26, 27, 28]. These systems provide functions that can be shared among different operators. They have demonstrated that they are adaptive to assumable changes.

Few systems based on software agents, however, exist for the service management layer. As we mentioned in Section 2.2.5, functions in the layer are needed to fulfill the requirements of individual network operators, and whose changes are not predictable. It is difficult for such requirements to be fulfilled by current software agents. The difficulty of the applications arises from the nature of or issues in software agents. The adaptability of a software agent is usually generated by a logic that is coded in their programs. This logic is designed by a human designer. If this designer does not predict all situations to which the software agent should be adaptive, the agent may not deal with a situation that has not been considered. Even if the agent can evolve, no one can know whether the outputs of the evolved agent are corrected. Therefore, the current agent technique may not be applied to the function design on which we focus in this thesis.

Knowledge-based systems (KBS) have also been applied to many areas in which they assist human activities. KBS is composed of two main parts, a knowledge base and an inference engine. A knowledge base stores and extracts data that represents knowledge properly. An inference engine solves a problem using data from outside of the system that contains the engine, and that stored in a knowledge base. KBS can change or enhance its behavior by modifying knowledge, so that it is adaptive.

Rule-based representations have become popular for the storage and manipulation of domain knowledge in KBS. Two important reasons for this popularity are as follows [29, 30].

1) The modularity of the rule-based framework
2) The ability to use knowledge stored as rules in a nonprocedural manner

Rule-based representations have been used in the area of network management. In this area, this technique is referred to as policy-based management or policy driven management

[31]. We call it policy-based management in this thesis. In this approach, policies on how to operate and manage networks are knowledge to be represented as data in a system. Expression of policies and a searching mechanism have the key to make a system adaptive to an environment in which user requirements are changeable, as well.

2.3.3 Standard products

There are many standards with regard to the following areas.

1) System architecture
2) Business processes
3) Management parameters
4) Information models
5) Protocols.

We have already described some of these in this thesis. The standards of system architecture give three aspects, functional, informational, and physical [9]. All aspects conform to the layer model described in Section 2.1.2. Business processes and their interactions for network management have been defined as mentioned in Section 2.1.3.

As for management parameters, they keep the heat on set standard parameters with regard to contracts [32, 33]. The items below are examples of parameters set by TMF.

a) Time to restore service
b) Time to repair
c) No service provider liability interval

The purpose of the information models is to give structure to information conveyed externally by protocols. In addition, it is to model management aspects of the related resources [34]. The information models are based on object-oriented analysis and design. Object classes are abstractions of data processing, or communication resources (e.g. connections and routers) for the purpose of management. There are technology-independent classes such as "System", "Log" [35], "Network", "Trail" [36], and "Basic Layer Network Domain" [37]. In addition, ITU-T (International Telecommunications Union – Telecom Standardization) provides many classes dependent on individual technologies, such as for SDH (Synchronous Digital Hierarchy) [38], ATM (Asynchronous Transfer Mode) [39]. TMF also provides object classes for wide areas, such as work in network management [40] and world ordering [41].

These object classes are usually dependent on specific protocols. Several protocols are currently recognized as the standards for network management. The major protocols are CORBA/IIOP (Internet Inter-ORB Protocol) [16] and SNMP [15]. We can see many NMSs

based on these two protocols [42, 43, 44, 45]. Java™ RMI [46] is getting affinity to COBRA/IIOP [47] so that we can make use of it instead of CORBA/IIOP. HTTP [48] is also expanding their applied areas [49, 50].

System designers should conform to some or all of these standards for cost reduction, short period for constructions, and/or interoperability of NMSs. On the other hand, they have to pay attention to user requirements that need original functions. Documents of these standards, however, make little mention of ways to add a new feature to, or modify items set in the standards, such as information models. Scant literature has given an account of this problem, either. Therefore, system designers have to consider by themselves how to bring unique functions into harmony with the standards.

2.3.4 Off-the-shelf components

In line with the progress of the standards for network management, vendors have been developing off-the-shelf components [51]. They usually provide functions that can be shared among different network operators. They keep interoperability between components from different vendors, by means of conforming to the standards and approval before use [52].

The use of these components lightens the workload of system designers. It may eliminate building an NMS from scratch. System designers may make use of off-the-shelf components as a part of an NMS to be built.

Almost all system designers, however, encounter the same problem as in the case of the use of the standard object classes. Off-the-shelf components may not provide proper methods to make functions specific to a network operator. If designers make use of off-the-shelf components in NMS construction, they have to use the components as black boxes, because few vendors open source codes in their components. Little literature has dealt with this point.

Chapter 3

Function design led by users

In this chapter, we will first describe concepts of the function design led by users (FDLU) that network operators take the initiative in eliciting and describing requirement. A survey of traditional techniques in RE follows and clarifies issues for use of FDLU. We will describe our technique in which use cases detailed by policies specify functions to be built. In the last of this chapter, we will show a case study to validate our technique.

3.1 Concepts of FDLU

FDLU must be user-friendly and practical for system construction. This section surveys user friendliness and practicality via studies published so far, and defines the concept of FDLU in this thesis.

3.1.1 User friendliness

The IEEE standard dictionary defines the terms "user friendly". The meaning of this term is "Pertaining to a computer system, device, program, or document designed with ease of use as a primary objective" [53]. User-friendly tools and techniques provide means that are easy to use for expressions and/or executions. If the tools and techniques involve user's viewpoints, users can easily understand them. Consideration of user's viewpoints embeds lines of users thought into the tools or techniques, and eliminates extra learning to get specific knowledge for use.

User-friendly tools are considered in the field of human-machine interface. For example, users can more effectively make programs for a sequence controller using a visual programming method with icons [54]. In modeling and analyzing networks, a tool hides all the mathematical details of models used for calculations of network performance [55]. Users of this tool do not need to set various parameters when they use this tool. Therefore, they do not have to know the complicated model itself.

Inclusion of user's viewpoints has been considered in order to develop a tool friendly to users. In the field of software quality, it has been recognized that user's viewpoints are different from ones for developers [56]. Software engineers traditionally relate software quality to the number of defects in the code. On the other hand, a user is interested in knowing how frequently the software fails during its execution and how severe the effects of such failures are.

In the field of CSCW (Computer Supported Cooperative Work), it is recognized as a problem that participants usually have different backgrounds and standpoints. The mutual understanding among the participants is hindered due to these differences. A framework furthers the mutual understanding by picking up and showing differences in their intentions in collaborative design [57]. This framework considers both the user's and the designer's viewpoints to detect and indicate the differences.

We can summarize that user friendliness requires hiding the knowledge outside the user domain, the employment of symbols familiar to users, and a consideration of user's viewpoints.

3.1.2 Practicality for system constructions

User friendliness does not always have a positive effect on tools and techniques. It sometimes makes the tools and techniques unsuitable for specialists like system designers, because details useful to them are hidden.

From evaluation items in literature, we can see which conditions must be fulfilled to make a technique practical. Of course, conditions to be fulfilled depend on the context of an application. We focus on system design in this thesis, and clarify conditions to make a technique practical by surveying the literature published so far.

As a result, the following three points are important for a practical technique to design software:

1) Reduction of ambiguity
2) Detail tuning
3) Iterative design.

Formal methods can reduce the ambiguity in specifications and provide a basis for verification later on [58]. They are used in the development of some applications, for example, computer communication networks [59] and E-commerce [60].

Detail tuning as used in this thesis means that a specialist is able to set variables and/or expression in a specification or software design. Formal methods mentioned above usually provide the means to manipulate details in a specification or software design. As another example, we can use a method in which object-oriented analysis combines with extended

Petri-Nets for distributed software design [61]. Unified Modeling Language (UML) [2] also provides powerful diagrams that show the static structure and dynamic behavior of software. UML is used to build various applications. However, detail tuning is not always consistent with user friendliness because it often needs expertise and skills relating to software.

The iterative design of software is recognized as an effective way to fulfill user requirements. It is adopted in many methodologies, for example, Rational Unified Process [6] and the spiral model [62]. On the other hand, the water flow model [63] does not provide a framework for iterative design and many problems of this approach have been pointed out.

3.1.3 Characteristics for users to design functions

The following points are summaries of the nature for FDLU in order to be friendly to users and practical in building a software system. These become design goals of FDLU.

1) It should hide details relating to function design.
2) It should employ expressions familiar to users.
3) It should include user's viewpoints.
4) It should **not** permit ambiguous expressions.
5) It should be able to be used in iterative processes for development.

3.2 Traditional approaches to requirement analysis and description

Requirement Engineering (RE) is composed of three major processes to elicit, describe, and confirm requirements [20]. Figure 11 shows an overview of these processes and relations between them.

Elicitation of user requirements provides knowledge of the subject domain in which a system to be built will be used. This knowledge is used to describe specifications of software to be programmed. The importance of this phase cannot be overemphasized. A software system could not provide proper functions if system developers do not have the right knowledge about the subject.

A description of requirements produces models that will be used for design in the following phases. A model explains user requirements in terms of functionality in the system to be built. An aspect of non-functionality, e.g. security or performance, is described in another model.

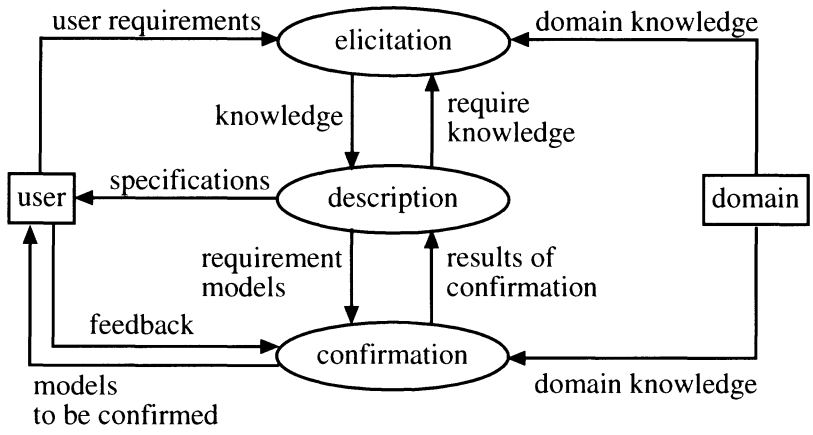


Figure 11: An overview of requirement engineering

In the process of confirmation, users come to an agreement with system developers in terms of requirements. For this agreement, users and system developers verify models made at the description stage. This gives feedback to the description process if the models are inconsistent, inaccurate, and/or ambiguous.

The way in which requirements are acquired and expressed is important for these processes. Many methods and tools have been proposed for the acquisition or modeling of requirements. We will review major methods and tools for requirement acquisition and modeling with regard to whether they have the user friendliness.

3.2.1 Requirement acquisition

This subsection describes several traditional approaches to acquire user requirements.

3.2.1.1 Approaches based on an interview

Interviewing users is an intuitive approach to acquire requirements to a system to be built. An open interview is regarded as the simplest interactions between an analyst and a user [64]. An analyst lets a user talk about his/her jobs in a free and relaxed style. The analyst attains a comprehensive knowledge of subject domain. We can say that this method includes the user's viewpoint since the user can talk about his/her jobs using familiar terms. However, it has some problems that the analyst cannot elicit detailed requirements and/or specify user jobs in detail.

A structured interview [65] is one of orderly approaches that overcome the

shortcomings of an open interview. An analyst directs the user's attention to a specific subject, and elicits the requirements that the analyst wants to know. The analyst poses questions based on viewpoints of analysts or system developers. Therefore, a structured interview does not always helps users to represent correctly their needs to functions.

3.2.1.2 Goal-oriented approaches

Goal-oriented RE is one of the distinct trends for eliciting requirements for an analysis of the wider context in which the system will operate. The rationale of developing a system is to be found from the environment in which the system works [66]. This RE is therefore concerned with the elicitation of high-level goals to be achieved by the envisaged system [66, 67, 68]. These goals are refined into specifications of services and constraints [67, 69, 70].

Practical experience shows that this approach has the following problems.

1) It is difficult for analyzers to find out goals [68, 71].
2) Breaking down a goal into composite goals cannot be carried out in a straightforward way, but is iterative [72].
3) It contains wasteful work in which uninteresting and spurious goals must be eliminated. This work, however, is difficult for analyzers [73].

These problems may be due to the lack of user's viewpoints, since almost all tools and techniques based on the goal-oriented approach stem from the viewpoints of analyzers. If user's viewpoints are included in a tool or technique, they might assist users to pinpoint their goals properly.

3.2.1.3 Scenario-based approaches

A scenario-based approach is also principal in RE and distinct from the goal-oriented one. In general, a scenario is a story that tells how a system satisfies user requirements. Scenarios have proved useful in requirement elicitation. They allow users and developers to envisage situations in which a software system to be built works, and assist in eliciting requirements from users [74, 75]. They also help to discover exceptional cases [71, 74, 76], or to derive conceptual models [77]. Some studies benefit from advantages of the goal-oriented approach and scenarios by combining them [72, 78].

One of the key points in this approach is how to produce a scenario. Most approaches have been designed for system developers. Therefore, the developers have to leans business, environment and/or context in which a system to be built works. It cannot be a solution to close the knowledge gap, since it sometimes takes a long time to learn. The specification

drawn up by users, however, might not be suitable for software development because few approaches included user's viewpoints.

As a rare case, Dano et al. adopted the "domain expert-oriented" approach [77], in which the domain experts can actively participate during the requirements acquisition activity by identifying and describing the use case. In this proposed method, tables with functions, conditions and assumptions are used to describe a use case, as shown in Section 3.2.2.1. If a method includes use viewpoints, users could actively take part in making a scenario.

3.2.1.4 Approaches with attention to multiple viewpoints

Viewpoints have been widely used in RE for a number of different reasons [79]. Primarily, its motivation has been the observation that different stakeholders will have different views and perceptions of the target domain. In addition, viewpoints have been used to characterize different classes of users [80], to distinguish between stakeholders terminologies [81], and to partition the requirement processes into loosely coupled pieces for work [82]. They have also been used to tolerate inconsistencies between development artifacts [83].

These approaches adopt multiple viewpoints, including users ones, to a system to be built. However, FDLU needs user's viewpoints for requirement acquisitions. Literature makes little mention of this point.

3.2.2 Notations for requirements

This part shows modeling languages in which user requirements are expressed. In general, requirements are composed of two parts, a functional part and a non-functional part. A functional requirement describes actions to be performed by a system. Physical constraints are not considered in this requirement. A non-functional requirement designates system properties that include constraints of environment or implementation, performance, maintainability, reliability, and/or scalability. We focus on the functional requirements that describe functions provided by a software system to be built.

We can see many proposals in which specification languages are defined. They are classified roughly into two categories. The first approach focuses on a static feature and describes that data is processed in software, e.g. Entity-Relationship (E-R) model [84]. The other focuses on system behavior and describes what happens when software executes (e.g. State Transition Diagram).

Object-oriented modeling has been incorporating these two approaches for a decade. Unified Modeling Language (UML) is the most widespread. It is set as a standard by Object Management Group (OMG) [2]. We review how applicable to FDLU this language is.

Languages for declaring policies are also reviewed in this part. Much literature has dealt with the languages in the area of the policy-based management.

3.2.2.1 UML

UML defines the following graphical diagrams.

1) Use case diagram
2) Class diagram
3) Behavior diagrams: state chart diagram, activity diagram, and interaction diagrams (sequence diagram and collaboration diagram)
4) Implementation diagrams (component diagram and deployment diagram)

A use case diagram is frequently used for a description of user requirements. Rational Unified Process (RUP) adopts a use case driven approach in which all diagrams are derived from a use case diagram, directly or indirectly [6].

A use case diagram is composed of actors and use cases, as shown in Figure 12. An actor inputs some data into and receives one or more results from a use case. Both a human use or computer system can be an actor. A use case diagram is easy to read and draw so that uses can make use of it.

As we mentioned above, other diagrams are derived from a use case. This derivation needs supplementary documents to present the specifics of a use case, because use case shows only a short comment. The specifics are usually written in a natural language such as English, so that they have a tendency to be vague. Some literature has proposed the use of another notation with formal syntax and semantics for the specifics.

Regnell et al. used Message Sequence Charts (MSCs) [85] to make formal

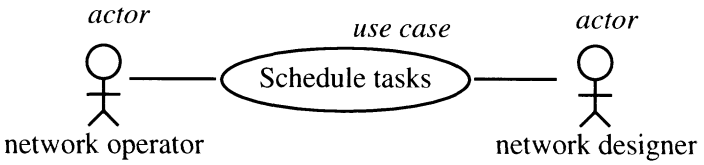


Figure 12: A use case and actors

specifications with use cases [86, 87, 88]. MSCs are frequently used to state the requirement for telecommunications software. Interactions between an actor and a use case are sequential in this proposed method. An MSC-based approach has advantages over the grammar-based approach, because it can be scalable and understandable.

Dano et al. adopted an approach in which domain experts can take part in requirements acquisitions [77], as written in Section 3.2.1.3. A domain expert uses an extended tabular notation to represent dynamic behavior of a system at the first setout, as described in Table 1. Secondly, an analyst produces Petri nets that show the behavior from the tabular notation. Even novices at designing software can write tables and show their requirements.

Constraints-based Modular Petri Nets (CMPNs) has been proposed to describe use cases formally [89]. CMPNs cover the shortcomings of existing Petri nets, such as place/transition nets (P/T nets) [90] and colored Petri nets (CP-Nets) [91], for formalizing use cases. The process in this approach starts with filling out an action-condition table for each use case. Therefore, users might actively take part in the process, though this approach does not have a great deal of interest in the participation of users. CMPNs itself, however, might be difficult for users to understand.

These methods mentioned above are promising for FDLU if users can draw up their usage of a system sequentially.

Table 1: The extended tabular notation to represent dynamic behavior [77]

FUNCTION Nber	FUNCTION	STATE OF OBJECT TYPES						CONDITION	ASSUMPTION
		PUMP	NOZZLE	TANK	PUMP DISPLAY	MOTOR	TRANSACTION		
$f_{1,begin}$	initiate the execution of the "take gas from a given pump" use case	UD	UD	UD	UD	UD	UD		
$f_{1,1}$	the customer removes the nozzle	idle	in	not empty	displaying an amount	on	UD	the pump is enabled	the pump is enabled
		idle	in	UK	displaying an amount	off	UD	the pump is disabled	
$f_{1,2}$	the customer pulls the trigger	initialising	released	not empty	initialising	on	UD		
		ready to deliver	released	not empty	displaying an amount	on	UD		
$f_{1,3}$	the customer releases the trigger	g gas	pulled	not empty	g amount	on	UD		
$f_{1,4}$	the customer replaces the nozzle	ready to deliver	released	not empty	displaying an amount	on	UD		
$f_{1,end}$	terminate the execution of the "take gas from a given pump" use case	idle	in	UK	displaying an amount	on	being created		
		idle	in	UK	displaying an amount	off	created		

3.2.2.2 Description of Policies

We can see many languages in which policies are described. Examples of the languages are written as follows.

1) Language for Netmon, which are a monitoring system [92].
2) Language for management of distributed systems [93].
3) A policy set for an access control [94].
4) Description for an access control [95].

For example, the language for Netmon consists of two parts, a policy rule and policy defined event. A policy rule offers a form "event **causes** action **if** condition," and a policy defined event does it "event **triggers** policy defined event **if** condition." Other languages offer similar forms.

3.3 Requirement description by users employing use cases detailed by policies

In this section, we set out our proposal about function design led by users for network management systems. The motivation of our proposal arises from the following two points. First, few traditional techniques can support FDLU. Second, we cannot adopt methods that postulate to define a sequence of a function, which are described in Section 3.2.2.1, as a method used in FDLU. This is caused by the difficulty of making a sequence of processing of functions, as mentioned in Section 2.2.4.

First, we describe our method in which the specifics of a use case are expressed using a set of policies. We also state kinds of policies and representations in which policies are written. Then, we illustrate how to make a use case and map policies onto it. We also describe the role of system developers.

3.3.1 Description of requirements: use case detailed by policies

We adopt a use case and policies as tools for users to draw up specifications to software to be made. The use case model is easy for users to understand, due to its simple expressions. It states what outputs the software provides for users, so that users can embed their viewpoints into a model. This user friendliness is underlined in the literature in Section 3.2.2.

A policy is a business rule for operators, and can be derived from their business goals. Policies in this thesis are classified into two categories. A policy in the first category shows a function and a condition by which the function is invoked. This type is referred to as a directive policy. A policy in the second category expresses the relationship between policies that are the first type. For example, a policy in the second category shows priority between two other policies. This type is referred to as a correlation policy. Section 3.3.2 will describe the details of each category and kind of policies.

Figure 13 depicts relationships between a use case and a set of policies, and ones between a policy and a function provided by a software component. Details of a use case are described by a set of policies. The number of policies and their relation are not limited so that users can decide a set of policies freely. A policy in the first category indicates one or more functions. When an NMS executes, conditions shown in policies are referred and policies whose condition are fulfilled are selected. The NMS performs a function indicated by a selected policy. The order of selected policies becomes a sequence of operations as a result. Therefore, a sequence of processing is decided dynamically and adaptive to state in a network or a service. This allows users to omit specifying a sequence of processing in a function.

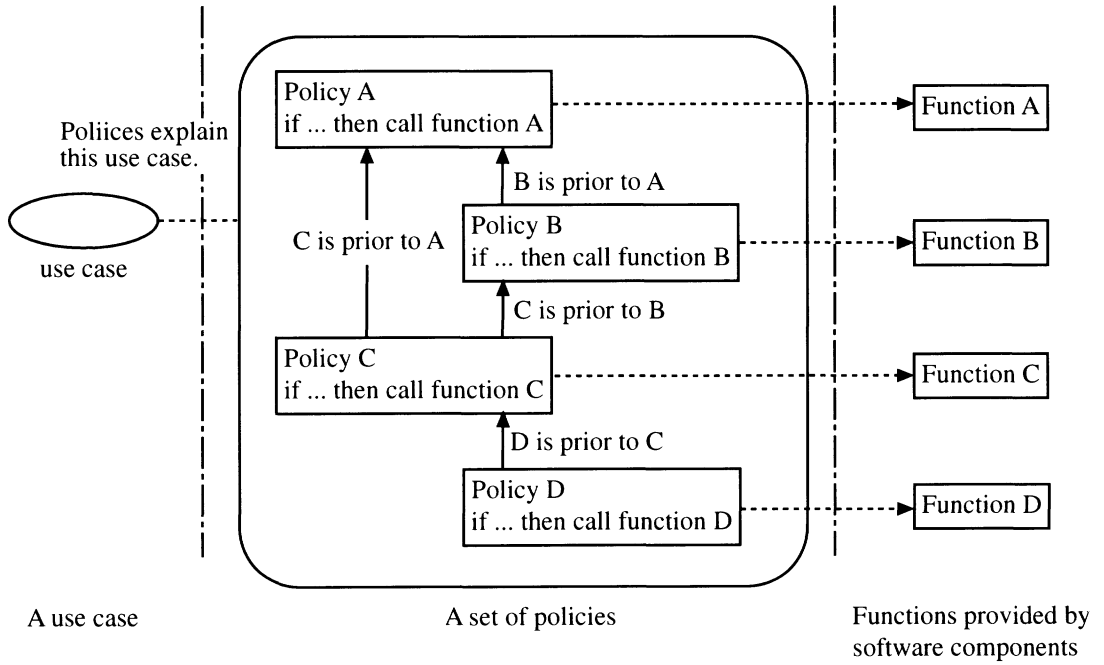


Figure 13: Relationship among a use case, a set of policies, and functions

This method combines features of the goal-oriented approach and the scenario-based approach. Policies are derived by manual analysis of an overall goal. In addition, the policy selection mechanism determines a scenario adaptive to a situation from a set of policies.

3.3.2 Policies

In this subsection, we define the kinds of policies and their relationships, and two types of representations in which policies are described. In addition, we describe how to add or delete a policy.

3.3.2.1 Kinds of policies and their relationships

It is important to offer proper kinds of policies, in order to specify user requirements. We have mentioned that there are two categories of policies for the FDLU. This method provides some kinds of policies in each category, as described in Figure 14. Using these six types of policies, users describe specifications that show activities of a system to be built.

A directive policy expresses an operation for setting parameters in an NMS, or monitors/controls networks to be managed. In addition, it states a condition in which the operation is performed. A directive policy is defined as either of the following two kinds of policies.

1) **Affirmative policy**: This policy expresses an operation to work actively and a condition in which the operation works. Therefore, the operation written in this

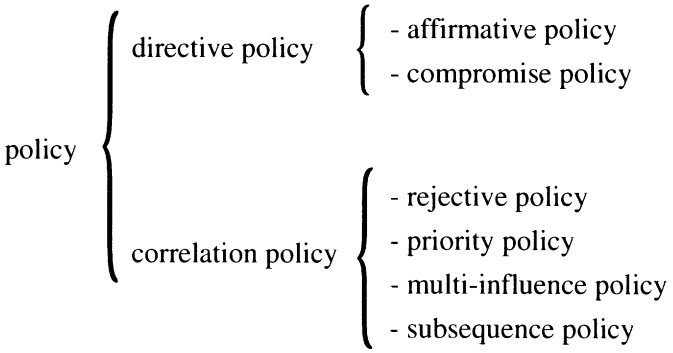


Figure 14: Kinds of policies

policy must be performed without any hesitation when the policy including it is selected.

2) **Compromise policy**: This policy expresses an operation that activates an affirmative policy that is suppressed by others. The affirmative policy is combined to another one by a rejective or a priority policy. If the compromise policy does not exist or it is not activated, the affirmative policy would not be selected. This policy contains expressions for specifying the affirmative policy to be supported. We offer two types of compromise policies in terms of activation of affirmative policies. Policy A is prior to policy B in the case illustrated in Figure 15. Policy A is activated and policy B is inactive when the conditions of both the policies are satisfied. The first type of compromise policy strikes a balance between mutually exclusive affirmative policies. When this type is introduced and adopted, it activates policy B in addition to policy A, as shown on the left side of Figure 15. The other type of compromise policy activates policy B that was inactive. Simultaneously, it inactivates policy A that was activated, as illustrated on the right side of Figure 15.

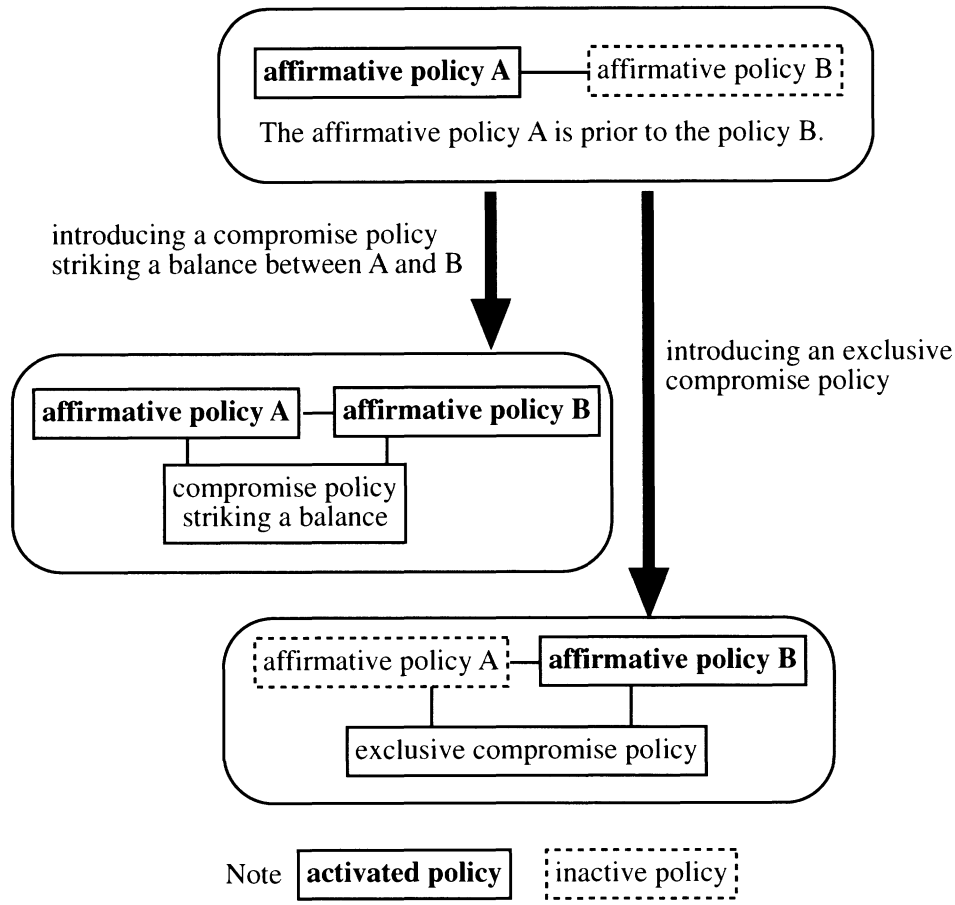


Figure 15: Two types of compromise policies

A correlation policy declares a relation between two or more directive policies, or between a condition and policies. We provide four kinds of policies described as follows.

1) **Rejective policy**: This policy declares a condition under which an operation must not work. In addition, it expresses the operation. This operation and condition must be written in an affirmative policy in advance. In addition, the condition declared in this policy must not be included in an affirmative one that specifies the operation declared in this policy.
2) **Priority policy**: This policy shows the priority between two operations. It declares these two operations as well. These two operations must be expressed in two different affirmative policies in advance. These policies can contain the same condition under which the operations work.

3) **Multi-influence policy**: This policy declares a condition that affects one or more affirmative policies. This condition must not be specified in the policies that are influenced by this policy. The effect of this policy is usually negative so that policies affected by this policy will be inactivated. The effect can be positive, however, it has the same meaning as an affirmative policy.

4) **Subsequence policy**: This policy shows an operation that follows another operation constantly. The expressed operations must be included in different affirmative policies in advance. This policy is not needed if the expressed operations are included in the same policy.

Note that each policy is composed atomic or composite elements, which mean an operation or condition. Figure 16 illustrates concepts of an atomic element and a composite one in this thesis. "Atomic" means that an element must not be a part of others as a whole or part. "Composite" means that an element is composed of atomic elements. A composite element cannot add new features to an atomic one. If users would like to make such an element, they have to define all atomic elements making up a composite one.

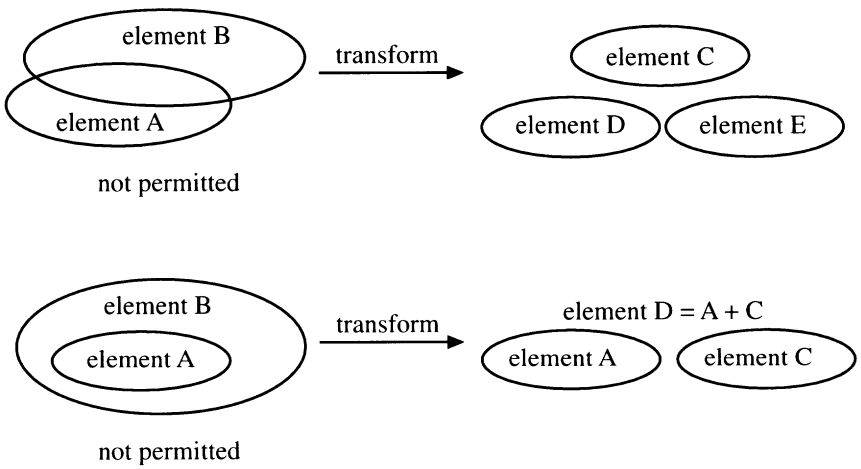


Figure 16: Transform from illegal elements to atomic elements

3.3.2.2 Representations of policies

It is important for FDLU to provide representations in which network operators can describe policies easily. We offer two types of representations, a graphical type and a table type. Network operators can use either of them depending on their preference. The graphical representation can be translated into the table representation, and vice versa, since both representations make use of the same meaning of policies.

Figure 17 shows affirmative policies represented in the graphical style. A rectangle with broken line shows a condition, and one with full line indicates an operation. An arrow means that a condition bound to the tail of the arrow activates an operation bound to the head of the arrow if a condition is fulfilled. There are five types of the affirmative policies. The simplest type is illustrated in Figure 17 (a). It means that condition A activates operation A if condition A is fulfilled. This representation provides a manner for describing logically AND using a notation shown in Figure 17 (b). A network operator can arbitrarily add a condition to

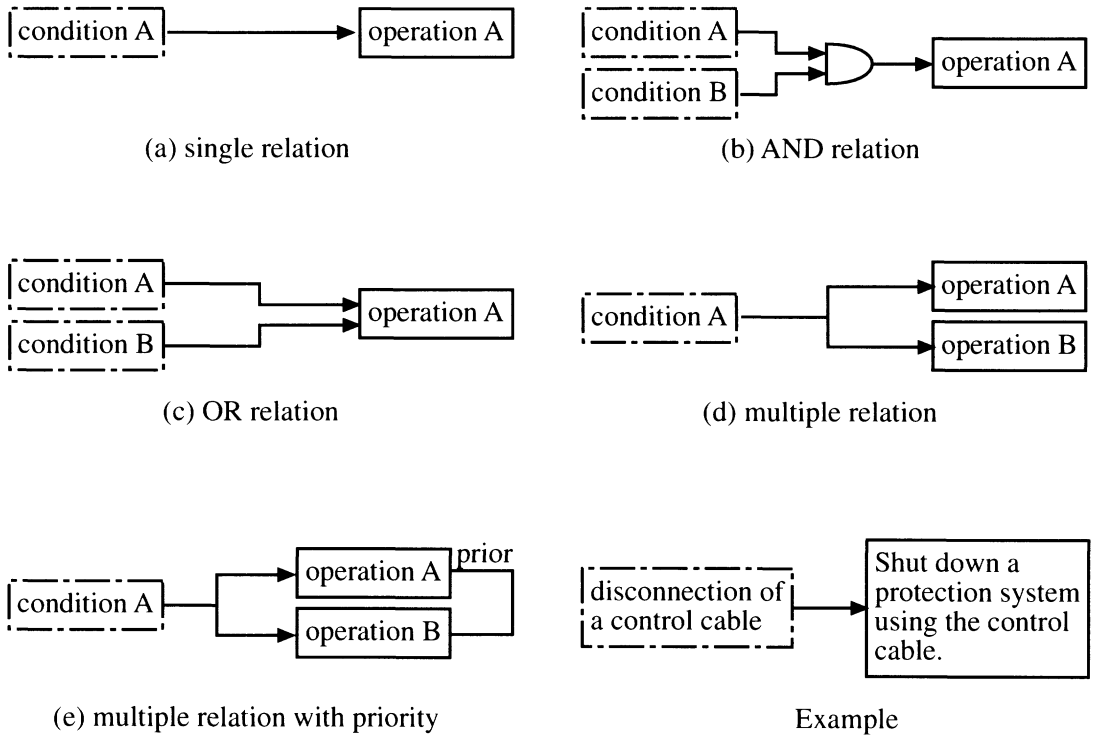
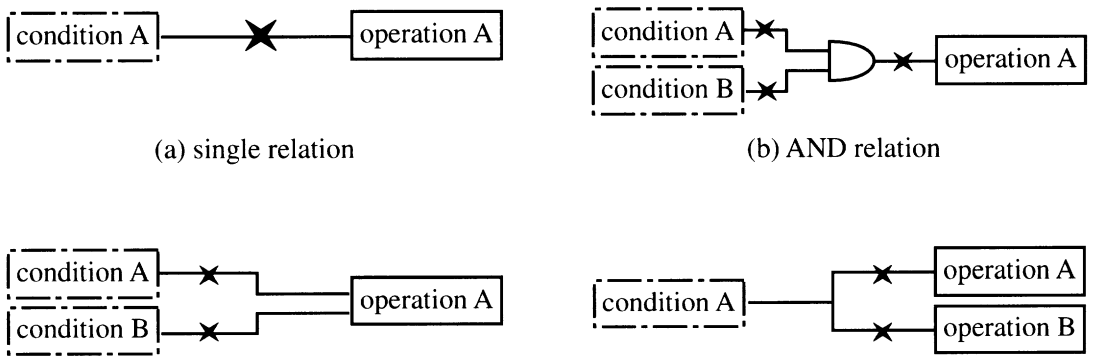


Figure 17: Affirmative policies represented in the graphical style

the AND symbol. On the other hand, logical OR can be written using single relations as illustrated in Figure 17 (c). Figure 17 (d) shows the case where operation A and B will be activated if condition A is fulfilled. If operation A is prior to operation B, a network operator has to set a prior relation shown in Figure 17 (e). These types can be combined to describe affirmative policies.

Figure 18 shows the graphical style representation for describing rejective policies. Rectangles have the same meaning as the case of the representation for affirmative policies. A line marked with an X binds an operation to a condition that has been defined in an affirmative policy. This line means the inactivation of the bound operation. Logical AND/OR are written in the same manner as for affirmative policies. The single relation of a rejective policy shown in Figure 18 (a) means that operation A cannot be executed if condition A is fulfilled.

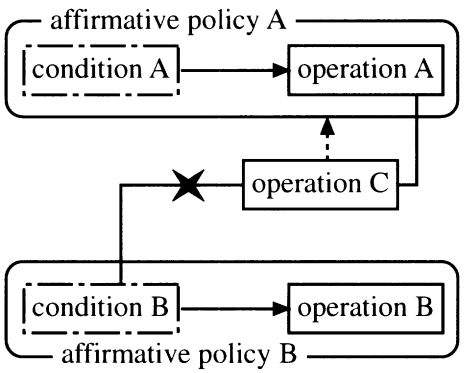


(a) single relation (b) AND relation

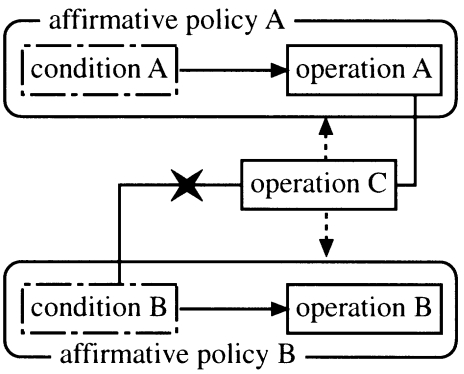
(c) OR relation (d) multiple relation

Figure 18: Rejective policies represented in the graphical style

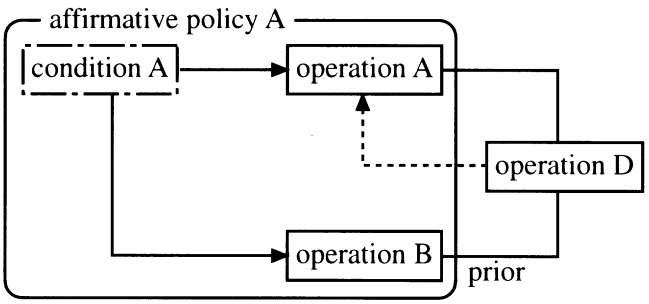
Figure 19 illustrates the graphical representation for describing compromise policies. An operation used in a compromise policy is put on the line that shows a rejective policy. This operation for the compromise policy has one or two arrows that indicate affirmative policies. In the case of an "exclusive type to a rejective policy" in Figure 19, the operation has only one arrow. Affirmative policy A pointed by the arrow will be adopted and affirmative policy B will not, if conditions A and B are fulfilled. In the case of the "balance type to a rejective policy" in Figure 19, affirmative policy A and B will be adopted under the same conditions. Figure 19 (c) shows a compromise policy related to a priority policy. This means that operation A is selected if the compromise policy is activated, and operation B is selected if not. A dotted arrow would be added to the graph if operation A and B are activated simultaneously.



(a) exclusive type to a rejective policy



(b) balance type to a rejective policy

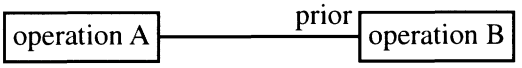


(c) exclusive type to a priority policy

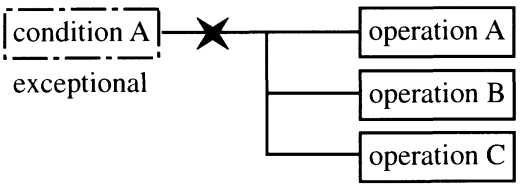
Figure 19: Compromise policies represented in the graphical style

Figure 20 shows the graphical type representation for describing other kinds of policies. For a priority policy, the keyword "prior" is set to an operation that should be prioritized. A multi-influence policy is written in manner similar to that for describing a reject policy. In this case, condition A has a keyword referred to as "exceptional", because this condition is not belonged to any affirmative policy. A subsequence policy is illustrated as an arrow that starts from an operation to a condition.

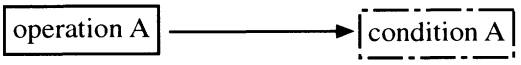
Network operators can also make use of table representations for the description of policies. Table 2 shows affirmative policies that are corresponding to ones described in Figure 17. Network operators can specify "AND" and "OR" in the column to indicate logical operators. This representation does not allow logical operators to be nested in order to keep table structure simple. If a condition activates some policies, the table can indicate a priority policy. The bottom row in Table 2 shows the operation A is prior to the operation B.



(a) A priority policy



(b) A multi-influence policy



(c) A subsequence policy

Figure 20: Other policies represented in the graphical style

Table 2: Affirmative policies represented in the table style

policy No.	Logical operator for conditions	Conditions	Logical operator for operations	Operations	Priority
1		condition A		operation A	
2	AND	condition A condition B		operation A	
3	OR	condition A condition B		operation A	
4		condition A	AND	operation A operation B	
5		condition A	OR	operation A operation B	prior

Table 3 is used to define rejective, priority, and compromise policies. Policy No.1, 2, and 3 in Table 3 reveal rejective policies that have been drawn in Figure 18. Policy No. 4 shows a priority policy, which is illustrated in Figure 20. If users set a compromise policy, they write it as policy No. 5, 6, or 7. These policies correspond to ones illustrated in Figure 19. Users have to define the compromise type in these cases.

Multi-influence policies and subsequence ones are simply written in other table representations. Table 4 depicts a table representation of the multi-influence policy that is shown in Figure 20 (b). The second column indicates a condition, and the last one does operations influenced by the condition. Users do not need to define whether the influence is positive or negative, because the condition written in the table always inactivates the operations.

Table 3: Compromise, priority, and rejective policies represented in the table style

policy No.	Restrained operation	Logical operator	Rejective conditions	Prior operation	Operation for compromise	Compromise type	Corresponding policy
1	operation A		condition A				(a) in Fig. 18
2	operation A	AND	condition A condition B				(b) in Fig. 18
3	operation A	OR	condition A condition B				(c) in Fig. 18
4	operation A operation B		condition A				(d) in Fig. 18
5	operation A			operation B			(a) in Fig. 20
6	operation A		condition B		operation C	exclusive	(a) in Fig. 19
7	operation A		condition B		operation C	striking a balance	(b) in Fig. 19
8	operation A			operation B	operation D	exclusive	(c) in Fig. 19

Table 5 shows a table representation of the subsequence policy illustrated in Figure 20 (c). The first column indicates an operation, and the second column states a condition that will be subsequent to the operation.

3.3.2.3 Addition or deletion of a policy

This section explains notandums for adding or deleting a policy.

When network operators add any kind of new policy using the graphical representation, they do not need to take a survey of the whole policy graph. They define a new policy and connect it to existing policies with which the new policy has a direct relationship. In addition, they confirm all relationships between the existing policies and the new policy, and change some of the relationships if needed. In the same manner, an existing policy in a set of policies may be changed or deleted.

When network operators add a new affirmative policy using the table representation, they can simply make a new row. When changing or deleting an affirmative policy, users need to confirm rejective, priority, and compromise policies that have direct relationships to the affirmative policy. The policies may be deleted if necessary. Users can also add, change or delete a multi-influence policy in this manner.

Table 4: Multi-influence policy represented in the table style

Policy No.	Condition	influenced operations
1	condition A	operation A operation B operation C

Table 5: Subsequence policy represented in the table style

Policy No.	Operation	Subsequcen conditions
1	operation A	condition A

When network operators add a new rejective, priority, compromise, or subsequence policy, they have to confirm the existence of affirmative policies that have relationship with the new policy. This is because a table defining the affirmative policies is different from that in which the new policy will be written. Other tasks to add a new policy are followed along with the pattern of the graphical representation. The change and deletion of these kinds of policy do not ask users to confirm the existence of an affirmative policy. Therefore, these procedures are same as that for the graphical representation.

3.3.3 Treatment of use cases

Use cases have to be treated in consideration of the notation using policies. Several techniques to create well formed use case models have already been proposed in literature. For example, these existing techniques have suggested that designers should limit the number of use cases [96, 97], and that they should create hierarchical use case diagrams optionally [98]. Our method reinforces the existing techniques in order to detail a use case with policies.

We have two concerns about the treatment of use cases detailed with policies. The first is whether a policy can appear in several use cases simultaneously, and the other is whether two policies have a relationship between use cases. To clarify these two concerns, we first illustrate associations between use cases that have been standards. Then we propose our strategies to treat use cases for detailed expressions with policies.

3.3.3.1 Standard associations between use cases

It is a good approach for drawing up a proper diagram to show a basic use case first, and then to expand it [96]. A basic use case is expanded by additional use cases, or by complementing the basic use case. There are two standard associations between use cases for the expansion [2].

1) **Extend**: An extend relationship from use case A to use case B indicates that an instance of use case B may be augmented (subject to particular conditions specified in the extension) by the behavior specified by A.
2) **Include**: An include relationship from use case C to use case D indicates that an instance of the use case C will also contain the behavior as specified by D. The behavior is included at the location defined in C.

Figure 21 shows a use case diagram employing these two associations. "Schedule maintenance tasks" is a basic use case in this diagram. This use case is expanded by another

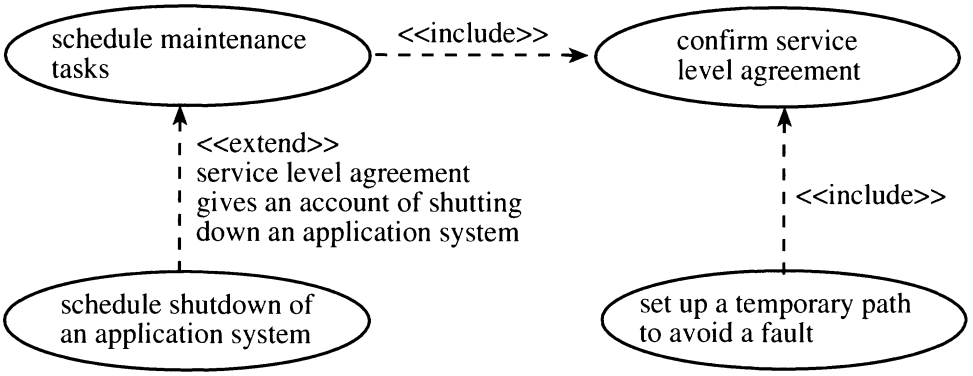


Figure 21: Standard associations between use cases

one named "Schedule shutdown of an application system," which works if a maintenance task requires halting an application system. The basic use case includes services defined in use case "Confirm service level agreement." Service level agreement is a contract in terms of services that a provider offers an end-user. This agreement is the principle for network management so that other use cases utilize this one.

3.3.3.2 Strategies to treat use cases

As mentioned above, we have concerns about the attachment of a policy to a use case, and the relationship between policies that are attached to different use cases.

Our strategy about the attachment does not allow that a policy belongs to multiple use cases. If a policy belongs to multiple use cases that have no relationship between them, a new use case should be derived from the existing use cases. The existing use cases have "include" associations to the new one to which the policy belongs, and remove the policy from the existing use cases.

If a policy belongs to multiple use cases that have an "include" relationship between them, it is necessary to see whether other use cases, so-called third-party use cases, have "include" relationships to the included use case. Only one of the positions remains in accordance with whether all the third-party use cases included it or not. Figure 22 illustrates a situation in which an affirmative policy belongs to use case A and B. In this case, use case C has an "include" association to use case B. The policy details use case A if use case C does not include the policy. On the other hand, it details use case B if use case C includes it.

If a policy belongs to use cases that have an "extend" association between them, it is not needed to confirm third-party use cases. We can imagine this situation if "extend" association between use case A and B is set in place of the "include" association in Figure 22.

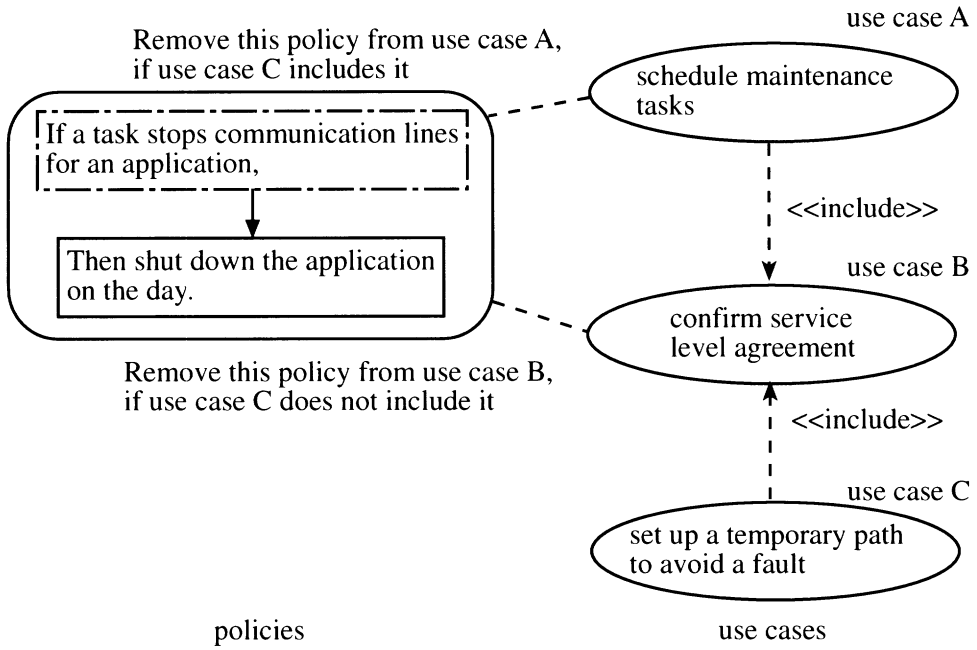


Figure 22: An arrangement of a policy in multiple use cases with include relationship

Only one of the positions can be selected due to the semantics of the use cases. In the case mentioned above, a user can decide that the policy details use case A or B, in accordance with only the meaning of the use case and the policy.

On the other hand, a relation between policies can cut across the boundaries of use cases in our strategy. Such a relation expresses a facet of an association of the use cases. Figure 23 shows a relation cutting across boundaries of use cases. There are two affirmative policies and a compromise one that has connections to the affirmative policy. The affirmative policy shown on the upper side in the figure details the use case that describes a function expressed as "schedule maintenance tasks." The use case on the lower side extends the use case on the upper side. It is detailed by an affirmative policy and a compromise one. This compromise policy must belong to the use case on the lower side, because one of the affirmative policy to which it connects details the use case. However, users should reduce the number of relations across boundaries for the sake of simplicity.

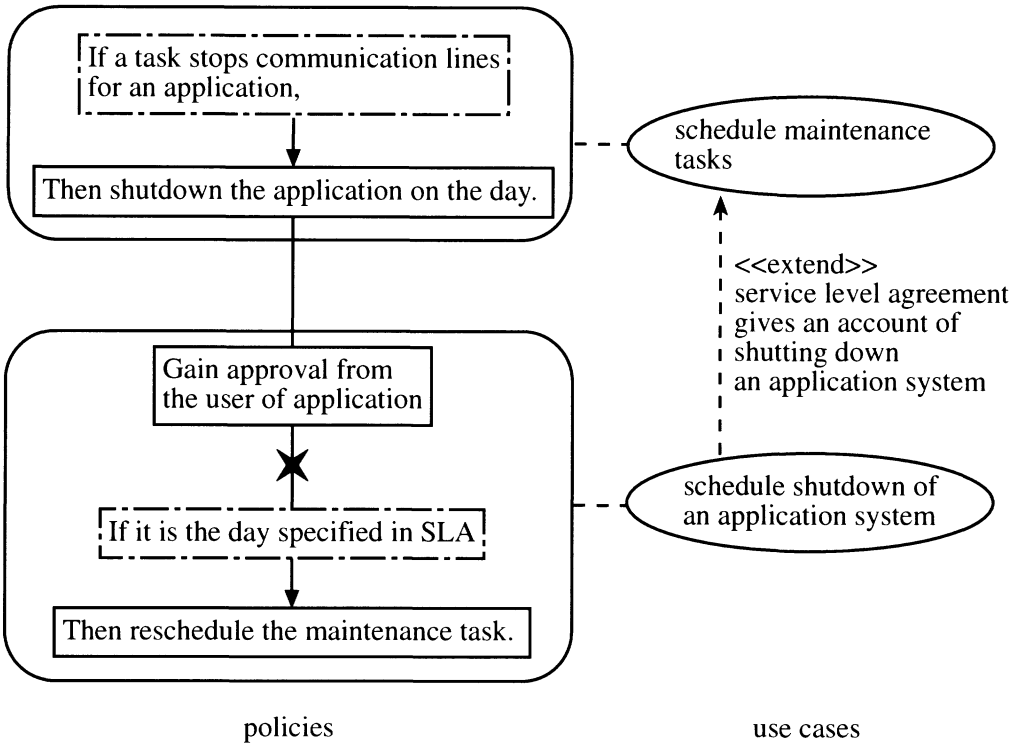


Figure 23: A relation of policies across the boundary of use cases with extend relationship

3.3.4 Roles of system developers

System developers coordinate an operation described in a policy and one or more functions provided by software components. Our method allows developers to make an arbitrary coordination between the operation and the functions in order not to constrain their activity. Therefore, developers may make small software components that provide each operation respectively. On the other hand, they may build a large component whose functions are identified by parameters in the function call. For NMS constructions, we recommend usage of the standard techniques and its expansions using our methods described in Chapter 5 and Chapter 6.

System developers may still have to analyze an operation written in a policy. This depends on maturity of description with policies. Developers make use of existing techniques, described in Section 3.2.

3.4 Case study

We applied our method to a design of NMS functions, in order to confirm its effect on requirements analysis and system design. In this section, we mention the overview of the case study and summarize feedback from users in terms of the ease of drawing up requirements. We also judge how easy it is to understand user requirements, and how useful for development it is.

3.4.1 Overview of the case study

Functions are to schedule tasks such as maintenance of network equipment in this case study. These tasks are scheduled in order to keep service level agreement and/or to comply with rules within a company. The network operators work for a company that provides communication services for power supply. Task planners in different branches set arbitrarily the date and time of a task. Service may be stopped or degraded by tasks because communication paths go through many sites. The network operators, therefore, gather information on tasks and schedule them so as not to inhibit services. The designed functions support the work of the network operators.

There are hundreds of rules for this scheduling, since provided services are various and many paths are shared between them. The network operators currently schedule tasks with a basic function that detects the need to shutdown an application. System developers have not ever been able to make advanced functions for the scheduling because the network operators could not explain the complex rules correctly and precisely. Of course, the developers must have needed to understand not only this scheduling but also the services and mission of the company. The volume of rules and context to be understood was also a factor in the reason that advanced functions have not appeared. This shows the limitation of the existing approaches described in Section 3.2.

We let the network operators design use cases detailed by policies in this case study. Figure 24 depicts one of the policy sets in the case study according to the graphical representation. This graph focuses on ensuring the performance of applications that monitor and control systems for power supply.

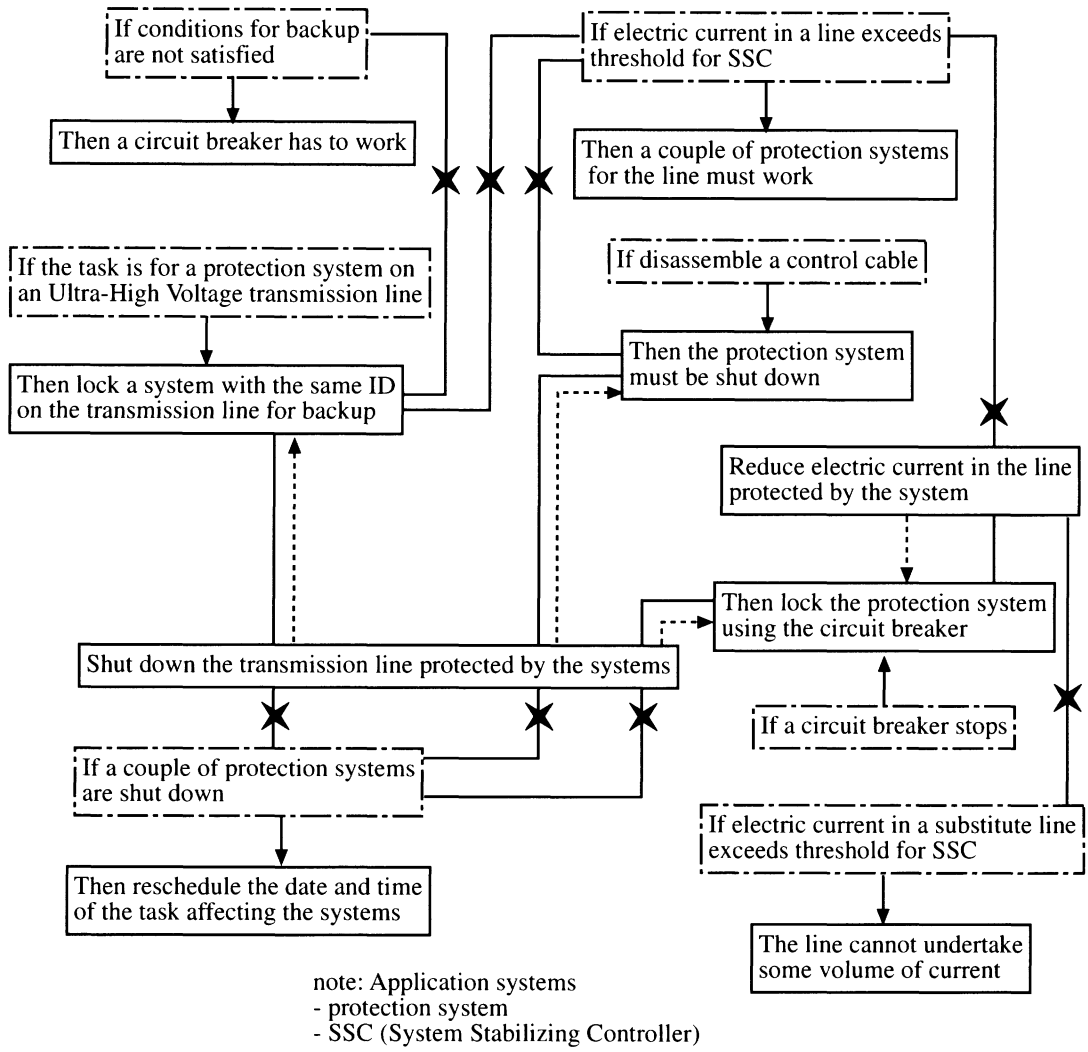


Figure 24: Policies developed in the case study

3.4.2 Feedback from network operators

Some operators preferred the graphical representation and others the table representation. This means that providing the two types of representations achieves a successful outcome. However, our method was inconvenient for operators who wished to use both of them. These two representations were not converted automatically since we had not supplied a CASE tool for the input of policies.

The proposed graphical representation has a low barrier for operators to describe their requirements. This style is familiar to the concept of their manuals, so that they did not deed to learn a large amount to draw a graph. We saw an advantage for the table representation as well, since they have used tables in their manual books. Allowing a natural language to be used also contributed to the usability of our method for the operators.

The operators spoke favorably about the semantics of policies. In particular, compromise policies were useful to draw up their requirements because they could not set priority statically. They did not require additional semantics of policies.

Use cases were understandable and useful for the operators to draw up. However, their products were not necessarily good, for example, the number of use cases was too high, or relationships between them were sometimes invalid. They should have learned the technique to overcome the pitfalls of use cases. The effort required to learn this technique seems to be the same as or less than learning other approaches for the network operators.

3.4.3 Feedback from system developers

Each policy described an operation and a condition in a natural language, so that the quality of description depended on the skills of the network operators to draw up requirements. If they conformed to the techniques to reduce the pitfalls of use cases and the strategies described in Section 3.3.3.2, policies assisted developers to make programs. If not, developers were puzzled as to how they should understand an operation, which was the common problem in RE.

Even if policies are not expressed appropriately, developers were endowed with the domain in which they work, because there were many standard and off-the-shelf components that could be used. They understood some operations written in policies using examples from these assets. Policies were useful in this scene since the operation was drawn up simply.

In terms of other operations, developers sometimes worked hard to make software components even though policies drew up operations simply. We saw some cases where more the two different operations written in different policies should have been provided by on component, even though network operators conformed to the strategies. Our method could not assist developers to solve the problem. It needed the traditional approach of RE and software engineering.

3.5 Summary of FDLU

Our contribution described in this chapter offers a new approach in which users lead the drawing up of function design. Such an approach has not ever appeared prior to this, since almost all techniques have focused on usefulness to system developers. Our method does not supersede traditional techniques in software engineering, but reinforces them.

Our method provides three techniques to facilitate function design led by users. These techniques enable users to draw up their requirements for functions by themselves. The techniques are,

1) Use cases detailed by policies
2) Policies suitable for network operations
3) Two representations for describing policies

We applied our method to a case study in which functions for scheduling tasks were developed. The result shows that our contribution confers benefits on requirement analysis, as mentioned below.

1) **User-friendly notations**: Users can draw up requirements for functions to be designed. This has not been provided in the existing approaches. This user-friendliness is supported by two techniques, described as follows.

    a) **Use case detailed by policies**: In the first stage, users make use case diagrams that show services provided by the functions to be constructed. A use case has a simple description so that it is friendly to users, even though the users are novices in function design. Second, users declare policies that detail a use case. A policy cannot be attached to multiple use cases. Our technique provides a solution to the case where a policy belongs to multiple use cases in the course of drawing up requirements. In this solution, users take account of the relation between the use cases to which a policy in question is attached. This is the same task to consider their own business so that the solution does not require users to understand a new approach.
    b) **Policies suitable for network operations**: A policy comes from a rule in network operations, so that it is friendly to network operators. In addition, we provide several kinds of policies that are more suitable for describing a rule in the operation. A compromise policy is most significant for network operations. This policy shows an operation that enables a prohibited operation to work. Of course, the operation mentioned in this policy is performed as little as

possible. This kind of policy has not been considered in the policy-based management, although it is often used in the real operations of networks.

2) **Facilitation of communications between users and designers**: Network operators can draw up their requirements that have not ever been passed to developers correctly. A policy declares an operation that shows one or more functions provided in an NMS. The notation of such an operation is simpler than that in a specification written in a natural language. In addition, the requirements are specified in a graphical or table representation. System designers can understand user requirements for functions more easily and correctly than before.
3) **High compatibility with object-oriented design**: Our method is based on use case diagrams. In addition, it does not modify techniques for system designers. Therefore, it is compatible with object-oriented design, especially RUP. System designers can adopt an iterative approach to design and implement the function that is specified based on our method. This feature is important for our method to enjoy wide acceptance.

Chapter 4
Policy selection

FDLU needs a mechanism to select a subset of policies from the total set defined by network operators. In this chapter, we describe a mechanism that is well-suited to the case of network management. We have adopted a concept of immune networks to the policy selection, in order to make the mechanism suitable for the domain.

First, we clarify the characteristics of policy selection on network management. Then, we review the related works and consider if they are suitable for the selection. In addition, we show a concept of immune networks applied to our mechanism. In the next section, we set out our proposal about the policy selection. We show the experiments of our mechanism and evaluate it after the proposal.

4.1 Characteristics of policy selection on network management

We have to take account of the following issues when building an NMS executing based on policies in accordance with the way mentioned in Chapter 3. Such NMS is referred to as a policy-based NMS hereafter. These issues come from the characteristics of network management as we mentioned in Chapter 2.

1) Claims to an NMS are different per network operator. Policies appear between dozens and thousands in a claim.
2) A policy states an operation that is invoked under a condition.
3) A subset of policies may be selected from the total set, because an NMS manages a number of networks and services.
4) Network operators can specify policies that conflict with each other. The conflict in this chapter means the case in which a policy forbids another one to be selected. A policy-selecting mechanism, however, must select policies that do not conflict with each other.

5) A temporary policy may be specified, or an existing policy may be changed for a certain period. In addition, networks or services that are managed vary in terms of their configurations or contents. A policy-based NMS, therefore, must treat policies or conditions that have not been assumed.
6) A policy-based NMS has to select a second best policy if the best one cannot be selected.

4.2 Related works

4.2.1 Works for policy selection

We can see the policy selection as a constraint satisfaction problem (CSP). CSPs have three components: variables, values, and constraints. The goal is to find all assignments of values to variables such that all constraints are satisfied. The classic N-queens problem is a typical instance of a CSP.

CSPs are NP complete so that no general algorithm for these problems exists. Algorithms for CSP are classified into two groups, exact ones and approximate ones [99]. Exact algorithms test all combinations of variables and values using backtracking, if needed. They guarantee the integrity, i.e. that solutions are found certainly if they exist, and the absence of solutions is confirmed. Exact algorithms have a problem with processing time although they find the optimal solution. A large number of policies are defined for network management. In addition, network operators need an output within a reasonable length of time. Exact algorithms, therefore, are ill-fitted to pick out policies in network management.

Approximate algorithms pursue to find a solution while they sacrifice the integrity. As for approximate algorithms, neural networks (NN) or genetic algorithms (GA) have been applied to various fields, and have shown their effects [100, 101]. The common feature of applied fields is that conditions for optimization do not vary, or elements to be processed such as a neuron in NN or a gene in GA are not added, deleted or changed. This feature is convenient for NN or GA, because of their characteristics as described in the following. In any type of NN, re-learning is required by a change of a unit or a relationship between units. In GA, a change in the domain causes a re-definition of the formula for fitness, and reconfigurations of crossover and/or mutation.

Recent literature has proposed re-learning in NN [102, 103] and flexible calculation of fitness in GA [104, 105]. A breakthrough in NN or GA was achieved by these contributions. Nevertheless, NN or GA can scarcely meet the requirements of the function design in network management. This is caused by an event in which a network operator defines new policies after NMS building.

4.2.2 A concept and applications of immune networks

A concept of immune networks has been applied to systems for information processing. This approach can also be classified to an approximate algorithm for CSP. Immune networks can well adapt to altering conditions, as well as being able to process various information [106, 107]. This concept, therefore, looks fit for the policy selection that deals with changes of network or service conditions, altering requirements, and processing of policies.

Two types of immune networks have been considered, as illustrated in Figure 25. These networks are composed of interactions in which an element such as a cell or molecular stimulates or suppresses another element. In both of these networks, stimulated elements come into force, and suppressed elements lose effect.

The first network is based on interactions in which T cells stimulate or suppress B cells. T cells and B cells are lymphocytes. B cells stimulated by T calls transform themselves into plasma cells that produce antibodies. Lymphocytes called helper T cell stimulate B cells when APCs (Antigen Presenting Cell) represent antigens to the T cell.

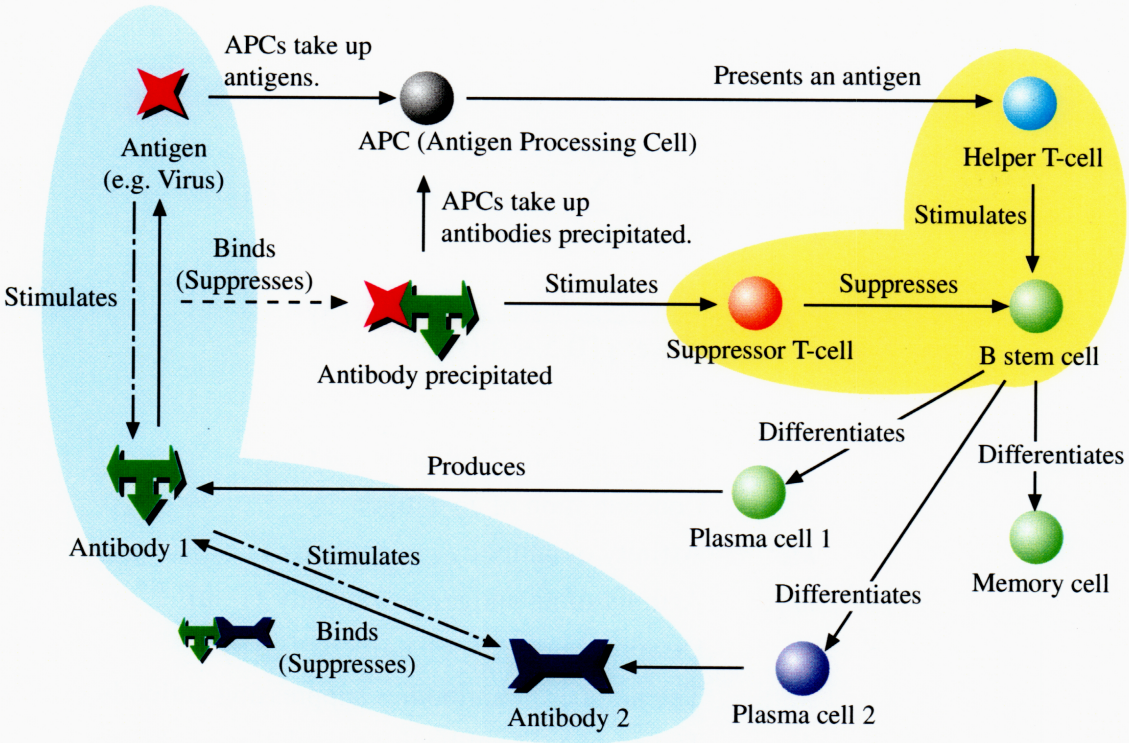


Figure 25: An Overview of Two Types of Immune Networks

The second network is based on stimulation and suppression between an antibody and an antigen, and between antibodies. This type of networks is referred to as idiotypic networks [108]. An antigen has an epitope that is a single antigenic determinant. Functionally, it is the portion of an antigen combining with an antibody paratope. An antibody consists of its main part, a paratope, and an idiotope. A paratope makes contact with an antigenic determinant. An idiotope is a single antigenic determinant in the variable domains of an antibody.

The idiotypic networks have been applied to a number of areas, for example, an autonomous mobile robot [109, 110], operations of a WWW server [111], and an agent supporting negotiation [112]. On the other hand, little literature has applied the first type of immune networks to engineering systems.

These systems based on the idiotypic networks find out a solution that is best suited to conditions, from options defined in advance. For example, the autonomous mobile robot grasps the positions of convenient objects (e.g. foods) and inconvenient ones (e.g. dangerous animals). Then, it decides its route based on the positions. The WWW server decides an action (e.g. providing a thread for each request) depending on conditions (e.g. size of a file to be transmitted). These adopt the formula (1) and (2).

$$\frac{dA_i(t)}{dt} = \left(\frac{1}{N} \sum_{j=1}^{N} m_{ji} \cdot a_j(t) - \frac{1}{M} \sum_{k=1}^{M} m_{ik} \cdot a_k(t) + m_i \cdot b - d_i \right) a_i(t) \quad \cdots(1)$$

$$a_i(t+1) = \frac{1}{1+\exp\left(0.5 - A_i(t)\right)} \quad \cdots(2)$$

$a_i(t)$: Concentration of antibody i at time t

b : Concentration of an antigen

m_{ji} : Affinity of antibody j to antibody i (≥ 0)

m_i : Affinity of an antigen to antibody i (≥ 0)

d_i : Disappearance rate of antibody i (≥ 0)

M : The number of antibodies suppressing antibody i

N : The number of antibodies stimulating antibody i

$A_i(t)$: The value for calculating $a_i(t+1)$
It does not have any meaning in the immune networks

Formula (1) reduces the effect of an antibody with high affinity and concentration if other antibodies have low affinities and concentrations. This nature has an effect on determining the relative merits of antibodies that are stimulated by antigens. This formula, therefore, is useful to select one policy from the total set. Besides, parameters in this formula have to be set for each application.

In the case of network management, policies that are well suited to conditions and compatible with each other must be selected from the total set in which policies have mutual relationships. It is impossible to make a solution, which is a set of policies, from policies that are selected one by one using the formula (1). This arises from the feature of the formula (1) that does not consider conflicts between policies stimulated by antigens.

4.3 Policy selection using artificial immune networks

This section set out our proposal about a mechanism to select policies. In this mechanism, we have applied the concept of the idiotypic networks to pick out policies well suited to conditions. The purpose of our proposal is to select a set of policies without conflicts, which have not been dealt with in literature so far.

First, we describe how each policy type described in Section 3.3.2 is attached to elements such as an antibody or an idiotope in immune networks. Secondly, we show how to select a set of policies.

4.3.1 Attachment of policies to immune networks

We have attached policies to elements in immune networks in accordance with features of policies, as illustrated in Figure 26 and Figure 27. The immune networks contain the concepts of the idiotypic networks and T-cells.

A condition written in a policy is mapped onto an antigen, as illustrated in Figure 26 (a). It is assumed that an antigen appears if the corresponding condition is fulfilled. The concentration of this antigen is valued as 1.0 in this case. The value of concentration is 0.0 when an antigen does not appear. Status of a service, network, or network element is reflected to an antigen in order to determine the value of concentration.

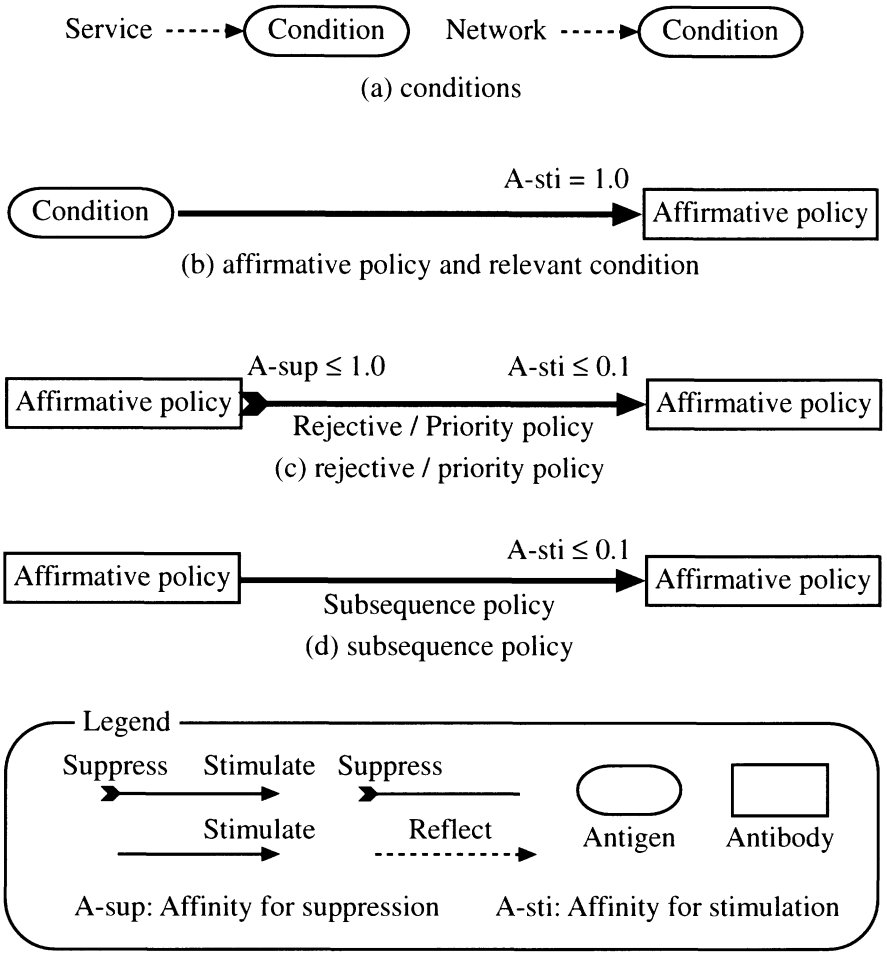
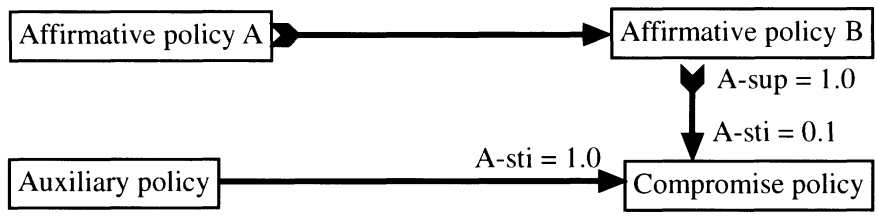


Figure 26: Relationships between policies and elements in artificial immune networks part I

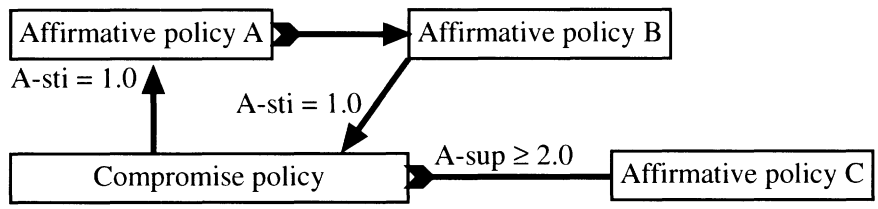
Each directive policy is mapped onto an antibody. The main part of an antibody represents an operation, and a paratope shows a condition due to which the operation is invoked, as well as the WWW server using an artificial immune network [111].

A correlation policy other than a multi-influence one is mapped onto an idiotope in an antibody. A rejective policy is represented as an idiotope that has bi-directional effects as shown in (b) of Figure 26. A priority policy is also done in the same manner. Note that the affinity for stimulation of these policies is determined upon rule #1 described in Section 4.3.2. As well, the affinity for suppression of these policies is determined upon rule #2 mentioned in Section 4.3.2. Figure 26 (c) shows that a subsequence policy is modeled as an idiotope only with stimulation, as mentioned in Section 4.3.2. This kind of idiotope relates an affirmative policy to another. Note that the affinity for stimulation of this policy is determined upon rule #1 written in Section 4.3.2.

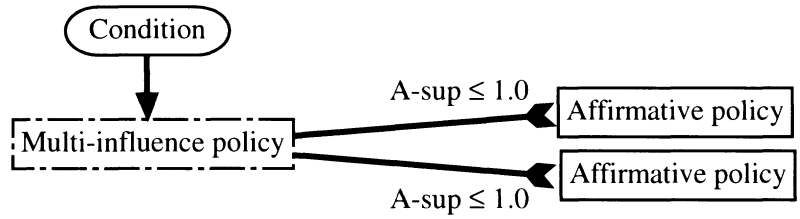
We have set two types of relationships between affirmative policies and a compromise one. These depend on whether the compromise policy is exclusive or not. The relationships shown in Figure 27 (a) have to be established for the case where a compromise policy is the exclusive



(a) compromise policy (exclusive type)



(b) compromise policy (balance type)



(c) A multi-influence policy as a suppressor T cell

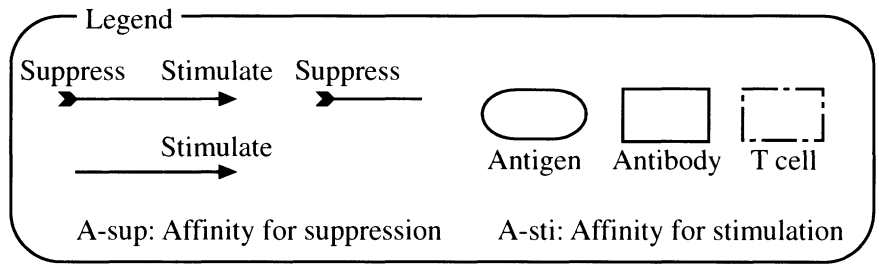


Figure 27: Relationships between policies and elements in artificial immune networks part II

type. In the case where both conditions for affirmative policy A and B are fulfilled, only affirmative policy B is selected if the compromise one is inactivated. In the same case, only affirmative policy A is selected if the compromise policy is activated. An auxiliary policy supports this selection in this case. It can be derived from the definitions of policies described in Section 3.3.2. It has the 'if clause' equal to that in the affirmative policy that would be selected if the compromise policy is activated. In Figure 27 (a), the auxiliary policy has the 'if clause' equal to the affirmative policy A. Therefore, it is stimulated by the condition that stimulates the affirmative policy A, as illustrated in Figure 28. It has an idiotope that stimulates the relevant compromise policy. If the affirmative policy A has an association in which another policy suppresses it, the auxiliary policy also has the same association to the suppressing policy, as illustrated in Figure 28. An auxiliary policy is not selected even if its concentration is over the threshold. This mechanism enables the compromise policy to be selected when the affirmative policy A and B are stimulated by the relevant conditions.

On the other hand, Figure 27 (b) shows the case where both affirmative policy A and B are selected, that is, the compromise policy is not exclusive. These polices are bound with idiotopes that have either a stimulating effect or a suppressing one. Another affirmative policy like the policy C in Figure 27 (b) can be set in order to suppress the compromise policy. Affinity for this suppression is set upon rule #3 that will be mentioned in Section 4.3.2.

A multi-influence policy is mapped onto a suppressor T cell as shown in Figure 27 (c). This is an application of the behavior of suppressor T cells to B cells. It can reinforce the concept of the idiotypic networks.

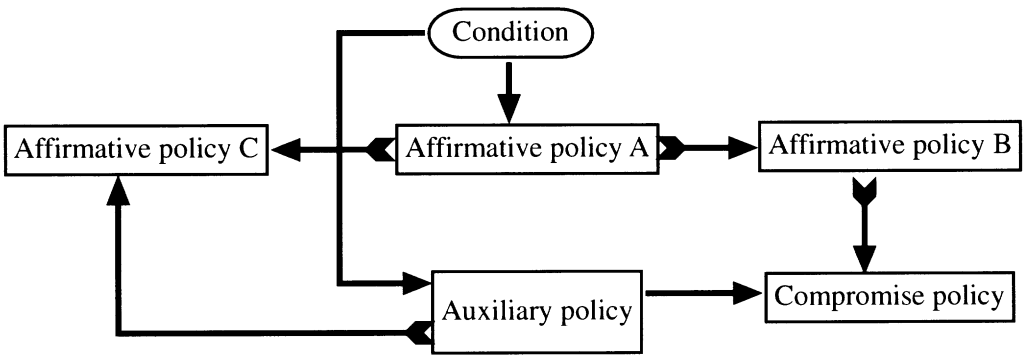


Figure 28: Relationships of an auxiliary policy to a condition and a suppressing policy

4.3.2 Selection of multiple policies without conflicts

We have introduced the following items into the artificial immune networks in order to select a set of policies without conflicts.

1) Introduction of asymmetric idiotope
2) Modification of the formula that calculates the concentration of antibodies
3) Introduction of rules for setting parameters

An idiotope used in conventional applications has applied the same value to affinity in terms of stimulation and suppression. We propose that the value of affinity for stimulation can be different from that for suppression in an asymmetric idiotope. It would not appear that the asymmetric idiotope is inconsistent to the concept of natural immune networks. It is not restricted to the influence increasing the concentration of antibodies stimulated being equal to one decreasing the concentration of the antibodies suppressed.

Antibodies are bound each other using the asymmetric idiotope only. An asymmetric idiotope sets the value of affinity for stimulation to 0 if it only suppresses an antibody with it. It sets the value of affinity for suppression to 0 if it only stimulates a relevant affinity. Idiotopes for subsequence policies and that are related to a compromise policy are set as mentioned above.

Conventional approaches have adopted the formula that makes use of the average of simulations or suppressions from antibodies, as mentioned in Section 4.2.2. Influences of antibodies are less than the influence of a relevant antigen using this formula. This feature is ill-suited to selection of multiple policies without conflicts. Only the existence of a relevant antigen has a key to whether a policy is selected. The formula does not consider influence from an antibody that causes a conflict.

The formula in our proposal makes use of the sum of influences from antibodies in order to remedy shortcomings for the selection of multiple policies. This amendment achieves a result in which influences from antibodies may exceed stimulation from an antigen.

Formula (3) calculates the gradient of concentration for each antibody in light of the sum of influences and effects from T cells. The formula (2) is used to calculate the concentration of each antibody based on the gradient.

$$\frac{dA_i(t)}{dt} = \left(\sum_{j=1}^{N_1} m_{ji} \cdot a_j(t) - \sum_{k=1}^{N_2} p_{ik} \cdot a_k(t) + m_i \cdot b + \sum_{l=1}^{N_3} n_{li} \cdot c_l - d_i \right) a_i(t) \quad \cdots(3)$$

$a_i(t)$: Concentration of antibody i at time t

b : Concentration of an antigen

m_{ji} : Affinity for stimulation of antibody j to antibody i (≥ 0)

p_{ij} : Affinity for suppression of antibody i to antibody j (≥ 0)

m_i : Affinity of an antigen to antibody i (≥ 0)

d_i : Disappearance rate of antibody i (≥ 0)

N_1 : The number of antibodies stimulating antibody i

N_2 : The number of antibodies suppressing antibody i

N_3 : The number of T - cells relevant to antibody i

We have defined rules for setting parameters such as affinity or disappearance rates, in consideration of the above scheme. Setting parameters has serious impacts on outputs from the artificial immune networks, which are a set of selected policies. We have set two guidelines described as follows.

1) An antibody is not selected unless it is stimulated by an antigen.
2) It is simplified to set values of affinity in antibodies that have not been adopted in the conventional approaches, such as a compromise policy.

Figure 26 and Figure 27 contain values to be set for each affinity.

4.3.3 Rule for setting parameters

Three rules for setting the values of affinity are illustrated in Figure 29. Rule #1 defines how to set the value of affinities in the case where an antibody is stimulated by others. The total value of affinity for stimulations is greater than 0.1, if stimulations from other policies to policy A increase the concentration of policy A without an antigen stimulating it. Besides, each value of affinity for stimulation is less than 0.1, if the concentration of policy A is not increased by only stimulation from policy B or C.

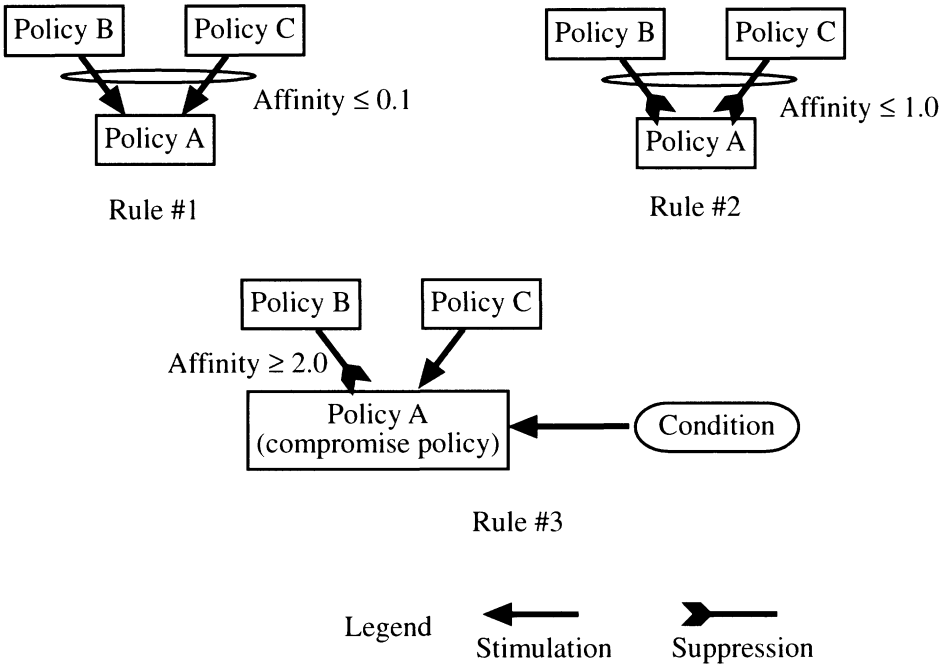


Figure 29: Parameter setting rules in cases where multiple policies affect a policy.

Rule #2 defines how to set the value of affinities in the case where an antibody is suppressed by others. The total value of affinities for suppression is not less than 1.0, if suppressions from policy B and C decrease the concentration of policy A. Besides, each value of affinity for suppression is less than 1.0, if the concentration of policy A is not decreased by suppression from only B or C. If stimulation from only policy B or C decreases the concentration of policy A, the value of the stimulation is not less than 1.0. If only one policy suppresses another policy, the affinity has to be set to 1.0.

Rule #3 regulates the value of suppression related to a compromise policy. Affinity for suppression from policy B is greater than the total value of affinities for stimulations to policy A. It is usually set over 2.0.

Besides, the disappearance rate of an antibody that means an affirmative policy is set to 0.1. The disappearance rate of an antibody that shows a compromise policy is set to 1.5.

The mechanisms have the features described above to calculate concentrations of antibodies, and pick out policies whose concentration is over a threshold. In addition, we have limited the range of A(t) in formula (2) from –5 to 5, in order to follow a change of conditions.

4.4 Experiments

We will explain experiments in which our method is evaluated. In addition, we will discuss applicability of our method to real operations.

4.4.1 Evaluation of proposed method

In order to evaluate the performance of the policy selection, we have made use of it in network operations for the quality of services. The networks to which the policy selection is applied provide communications to protect, supervise and control power systems. The service provider has contracts with users in terms of the quality of services. The contracts contain demands that are different from ones for public communications. A large number of conditions must be considered for the operations that fulfill the demands. The network operator, therefore, has to outfit them with functions that support these operations in light of various conditions. This feature seems to be well suited for the evaluation of the policy selection.

Figure 30 shows a typical model for these evaluations. We explain the results of these evaluations along with this model. This model shows policies that maintain a quality of services for CR (career relay) systems and SSC (system stabilizing controller), which are used for the protection or control of power systems. For example, policy no. 3 in Figure 30 means that an NMS has to keep the all-relevant CR systems working if a current on a line is over the threshold for SSC. Policies in this model are used in the real operations of a power company.

The policy and relationship with the dotted line in Figure 30 were used in an experiment in terms of the addition or deletion of a policy. Other policies and relationships were used in the all experiments.

A policy was written in a natural language. We used these letters as an ID of a policy without any language processing.

First, we show the validations of the asymmetric idiotopes and the proposed formula calculating the gradient of an antibody's concentration. Second, we explain that a reasonable set of policies was selected using our method with compromise policies. We also show that it is easy to add or delete a policy and the outputs at that time were reasonable for network operators.

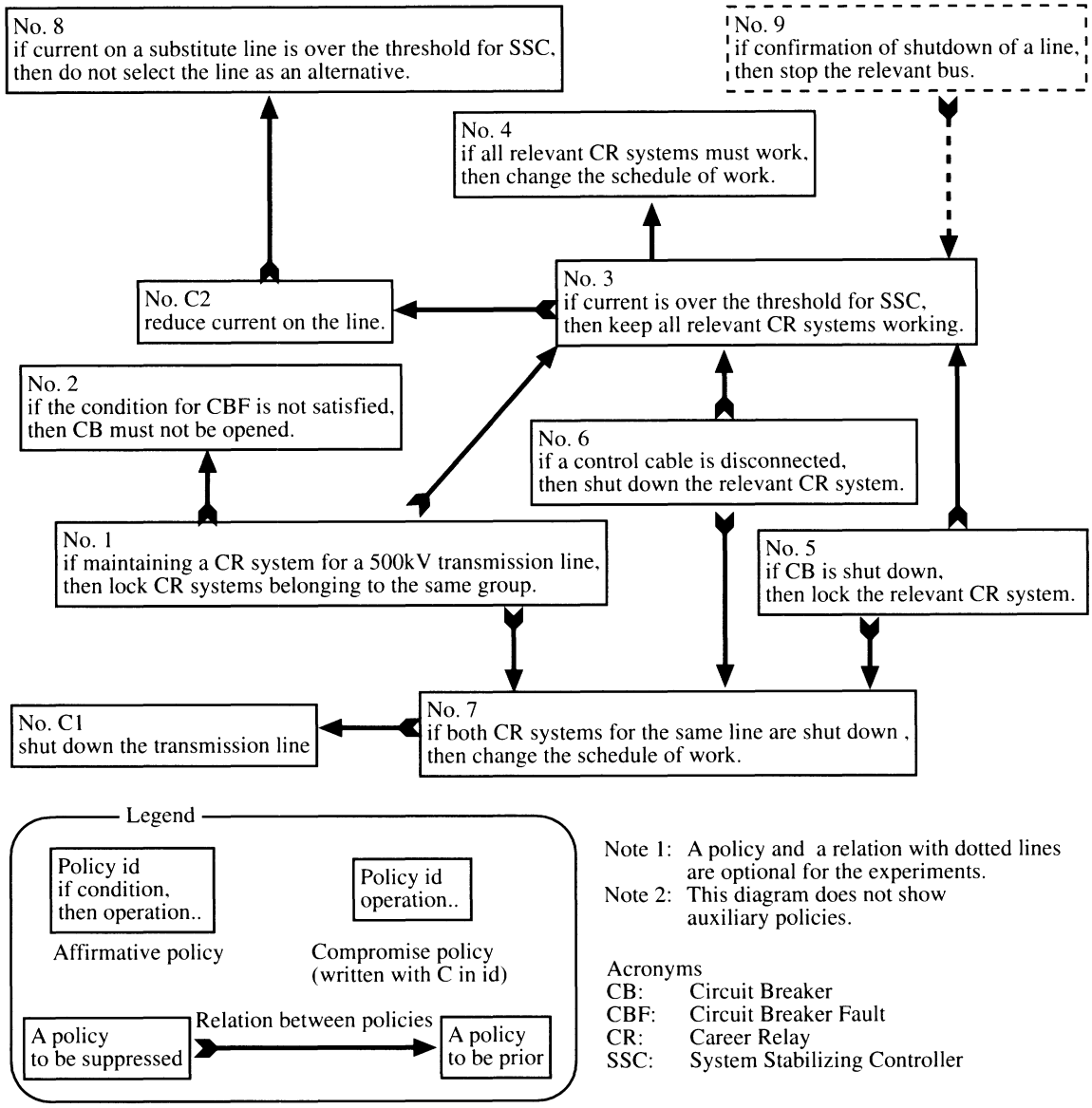


Figure 30: Policies for the evaluations of the proposed policy selection

4.4.1.1 Evaluations of Formula Calculating Concentrations

We compare the effects of the proposed formula (3) with ones from the conventional formula (1). In these experiments, policies were selected using each formula that made use of conventional idiotopes or asymmetric ones. We set all the values of affinity in the conventional idiotopes to 0.1. On the other hand, we set the values of affinity in the asymmetric idiotopes

according to the rules in Figure 29. All disappearance rates were also set according to the rules in Figure 29.

The formula (2) was used in every experiment, in order to calculate the concentration of each antibody. We set the threshold for policy selection to 0.7.

In the first experiment, it was assumed that the conditions described as follows appeared at the same time. We refer to this set of conditions as **condition pattern I**.

1) There would be work for a 500kV CR system.
2) Conditions for CBF would **not** be satisfied.

We have gained concentrations of antibodies in several cases under the condition pattern I. Figure 31 shows the results of the policy model using the conventional idiotopes, in the case where the conventional formula is used to figure out concentrations. Figure 32 shows the results of the model using the asymmetric idiotopes, with the same formula. Figure 33 shows the results of the model using the conventional idiotopes, with the proposed formula. Figure 34 shows the results of the model using the asymmetric idiotopes, with the proposed formula.

It took 2.71 seconds at an average rate of 1,000 steps on HP workstation K250. These processing times were independent of the formulas or idiotopes. In addition, it took almost same time for the processing in the case of experiments that will be described below.

These experiments indicate that a set of policies selected were not reasonable for the network operations in the case where the conventional formula was used. Concentrations of some antibodies shown in Figure 31 varied cyclically so that we could not pick out a set of policies. This defect is caused by the introduction of a compromise policy. A compromise policy forms a loop in the relationship antibodies. The conventional formula does not support such a loop structure in immune networks. In the experiment shown in Figure 32, policy no. 1 and 2 were selected. This pair, however, is against the correlation policy that the network operator has set between the policy no. 1 and 2. This defect is caused by a feature of the conventional formula. The formula takes the average of influences from antibodies, so that a policy stimulated by an antigen can be selected.

On the other hand, the experiments using the proposed formula exhibit that selected policies were suitable regardless of the types of idiotope (Figure 33 and Figure 34). This result means that the formula is fit to select multiple policies without conflicts.

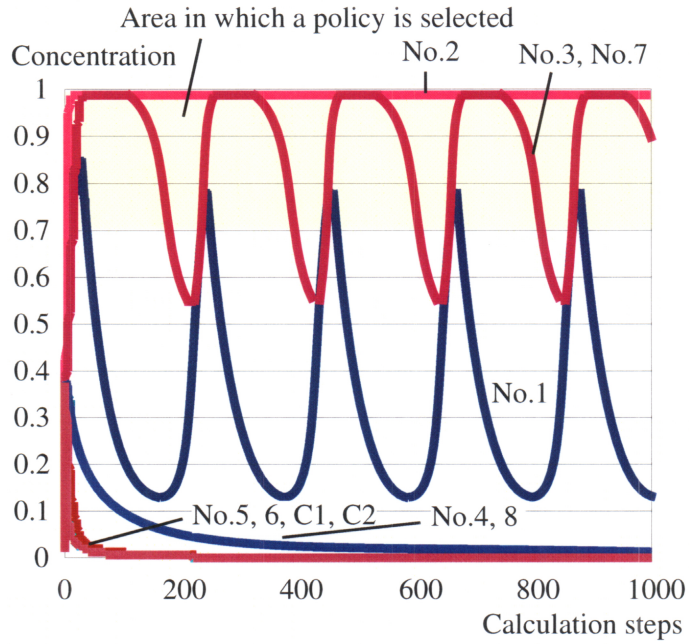


Figure 31: Transition of concentrations
(Conventional formula, conventional idiotopes, and condition pattern I)

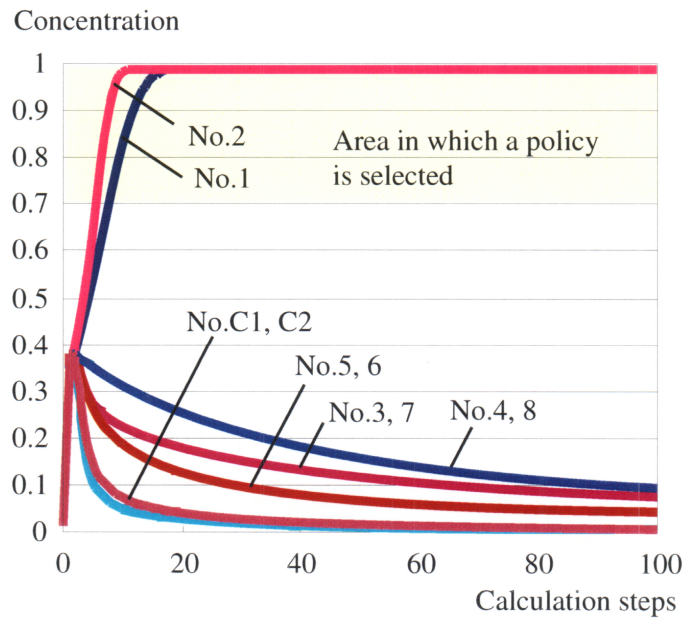


Figure 32: Transition of concentrations
(Conventional formula, asymmetric idiotopes, and condition pattern I)

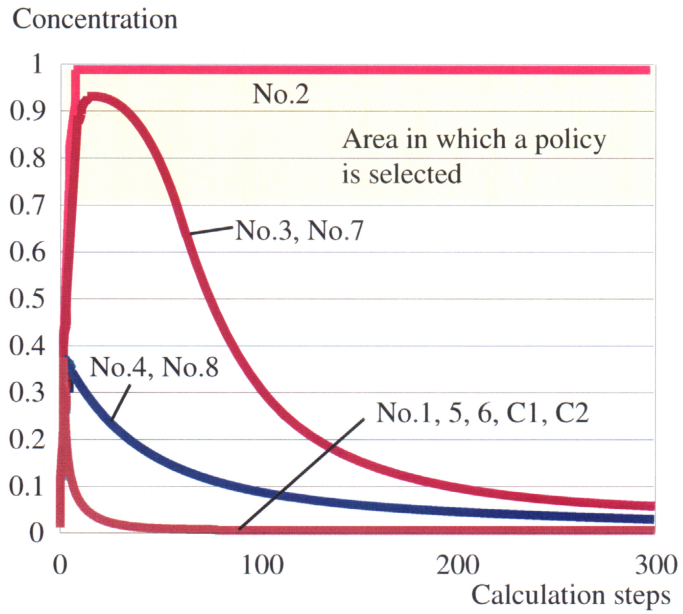


Figure 33: Transition of concentrations
(Proposed formula, conventional idiotopes, and condition pattern I)

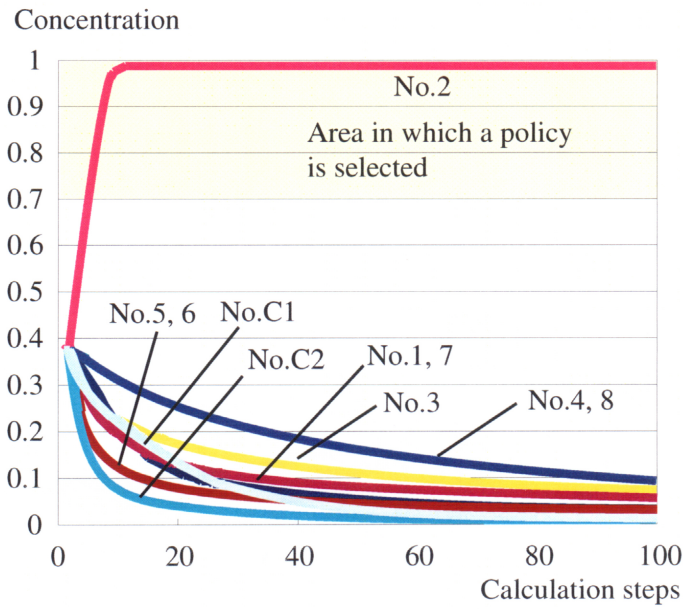


Figure 34: Transition of concentrations
(Proposed formula, asymmetric idiotopes, and condition pattern I)

4.4.1.2 Evaluations of the asymmetric idiotopes

We assumed **condition pattern II** in which the following two conditions appeared.

1) There would be work for a 500kV CR system.
2) Both CR systems for the same line are shut down.

In these experiments, formulas (2) and (3) were used to figure out the concentrations of antibodies.

Figure 35 shows the results of an experiment in which conventional idiotopes were used. The values of all affinity were set to 0.1. On the other hand, Figure 36 shows the results of an experiment in which the asymmetric idiotopes were used.

Policy no. 2 and 7 were selected in the results shown in Figure 35. They, however, are not consistent the conditions that were assumed to appear. This defect comes from the existence of an auxiliary policy. The auxiliary policy created for the compromise policy no. C1 stimulated policy no.2 and 7, so that the concentrations of these policies increased.

In the experiment shown in Figure 36, policy no. 1 did not increase its concentration in the opening steps of calculations, by suppression from policy no .7. The policy no. C1, however, achieved its effect on policy no. 1 in a matter of time. As a result, policy no. 1 was selected. This result is reasonable to the network operations.

From these results, we can conclude that the proposed mechanism can select a set of policies suitable for the operator's intention in the case where the total set contains compromise policies.

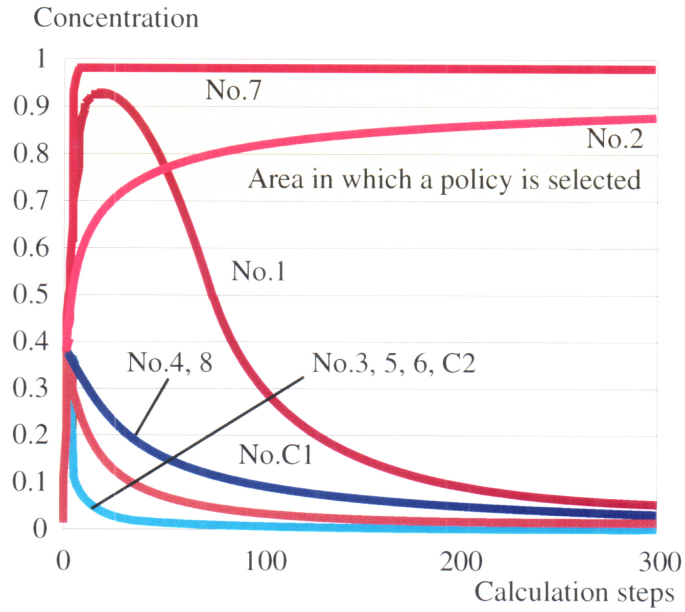


Figure 35: Transition of concentrations
(Proposed formula, conventional idiotopes, and condition pattern II)

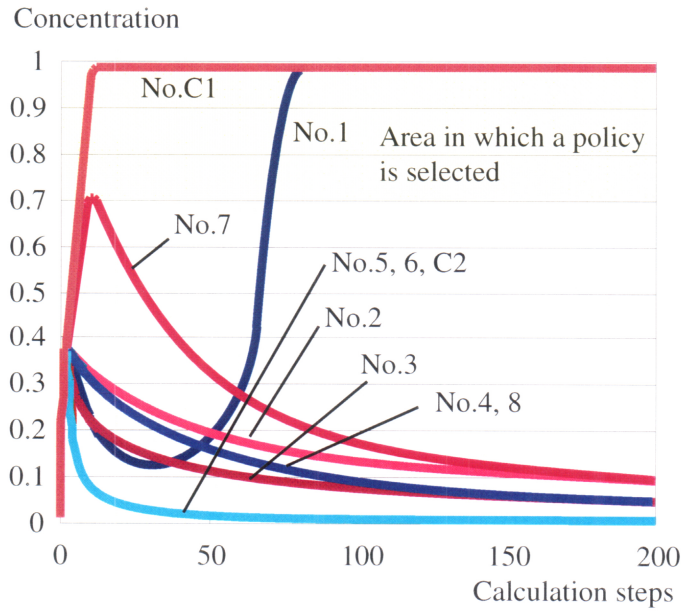


Figure 36: Transition of concentrations
(Proposed formula, asymmetric idiotopes, and condition pattern II)

4.4.1.3 Evaluation of Compromise Policies

We have shown the feasibility of our proposed method for selecting a set of policies without conflicts, as illustrated in Figure 36. We have carried out an experiment in another case. In this case, an affirmative policy suppresses another that is stimulated by a compromise policy. The compromise policy must not be selected because the affirmative policy it stimulates is not selected.

The following items are assumed conditions. Hereafter, we refer to this as **condition pattern III**.

1) There would be work for a 500kV CR system.
2) Both CR systems for the same line are shut down.
3) The condition for CBF is not satisfied.

Figure 37 shows the results of the experiment under the condition pattern III. Policies no. 2 and 7 were selected in this experiment. Policy no. 1 was not selected because the concentration of policy no. 2 suppressing policy no. 1 was high. Due to this situation, it was not necessary to select policy no. C1. Our method recognized this situation and reduced the

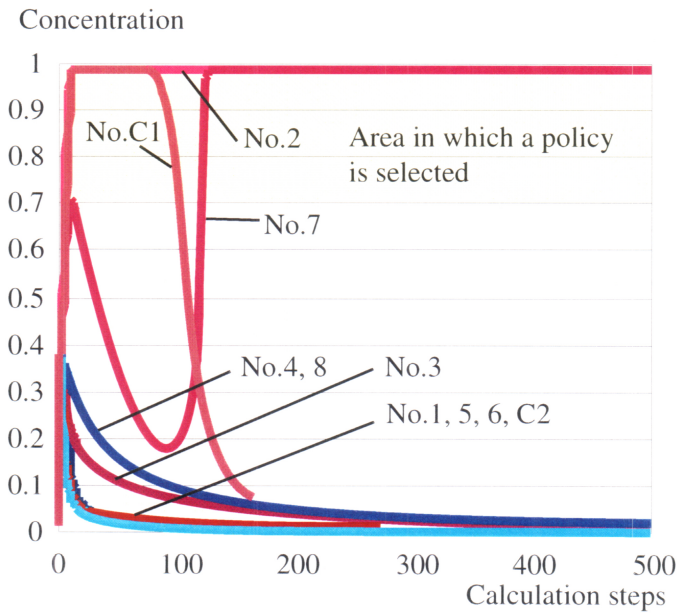


Figure 37: Transition of concentrations
(Proposed formula, asymmetric idiotopes, and condition pattern III)

concentration of policy no. C1. As a result, policy no. 7 increased its concentration enough to be selected.

These results show that a compromise policy is selected if it is needed and allowed. In addition, the existence of a compromise policy does not require a change of affirmative policies.

4.4.1.4 Evaluations of addition and deletion of policies

The addition of a new policy requires only definition of the new policy and reconfiguration of policies that have direct relationships to the new one. It does not need change of the formulas or rules for setting parameters. Figure 38 illustrates the procedure to add policy no. 9 in the experiments. The new policy was defined in the first step. We looked over policies that seemed to have relationships to the new policy, and set up idiotopes in the second step. At this time, we reconfigured parameters according to the proposed rules. In the third step, we created an auxiliary policy because policy no. 9 had a relationship to a compromise policy no. C2.

Our method for selecting policies was carried out after the addition of policy no. 9. In the case of each condition pattern, it selected the same policies as shown in the previous experiment corresponding to the same case. We have carried out other experiments in which policy no. 9 was stimulated by an antigen. These results are given in Table 6. The network operator that has defined the policies has deemed these results appropriate.

Neither did deletion of an existing policy need changes of the formulas or rules for setting parameters. It required only the tasks described as follows.

1) Identification of a policy to be deleted.
2) Dissolution of relationships that the policy has.
3) Reconfiguration of parameters according to the rules.

These results reveal that our method can add a new policy or delete an existing one without changing any part of the artificial immune network (e.g. formulas or rules for setting parameters). We can therefore conclude that our method is more suitable for the network operations than neural networks in which re-learning is needed for a change, or genetic algorithms in which formula and/or crossover must be reconfigured for a change.

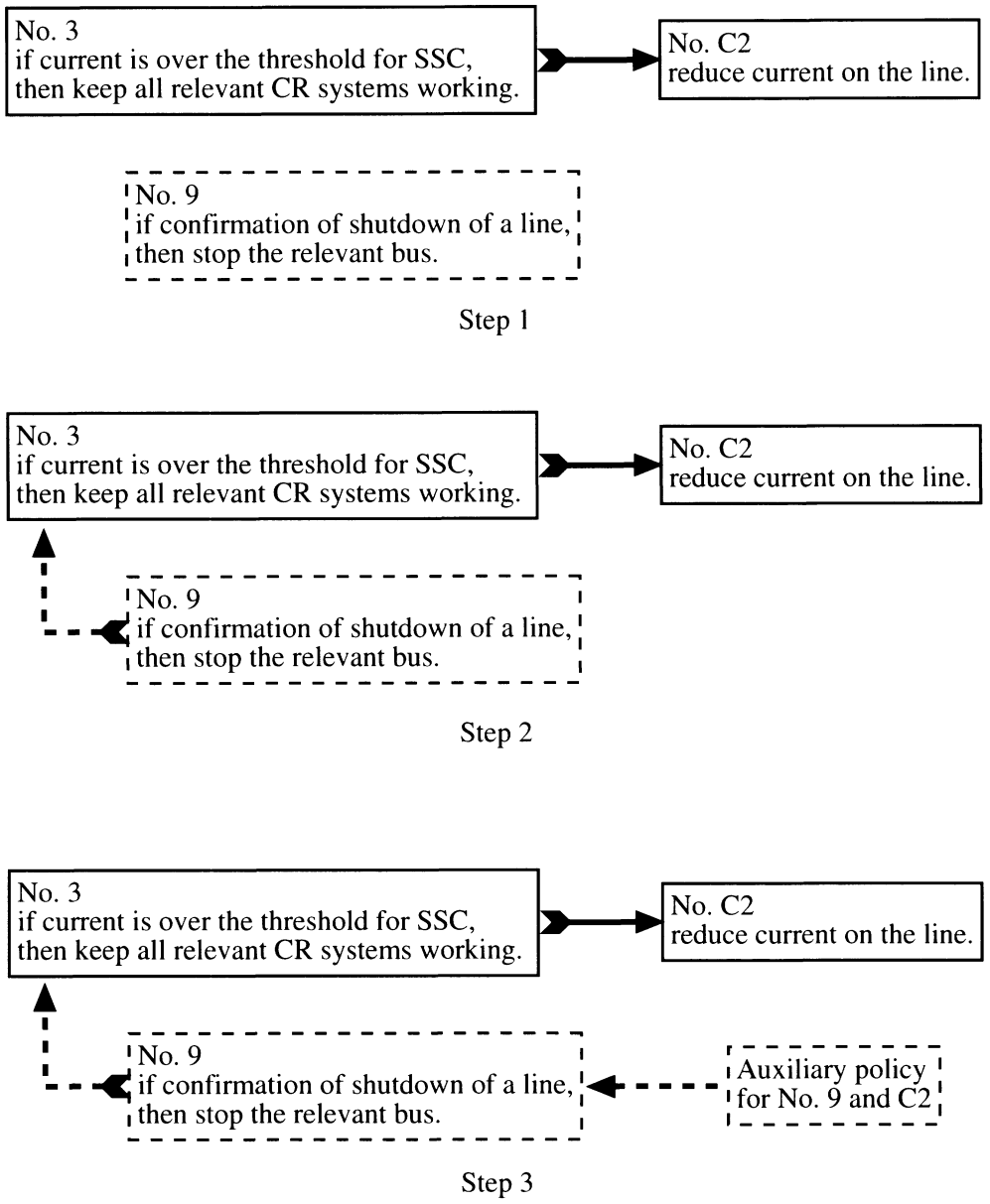


Figure 38: Procedure to add a new policy to existing ones

Table 6: Policy Selections after no. 9 is added

condition pattern	occurrence	selected policies
pattern IV	confirmation of shutdown of a line	No. 9
pattern V	confirmation of shutdown of a line current is over the threshold for SSC	No. 9 No. C2
pattern VI	confirmation of shutdown of a line current is over the threshold for SSC current on a substitute line is over the threshold for SSC	No. 3 No. 8

4.4.2 Consideration of application to the real operations

We will validate the results of the experiments in terms of application to the real operations. In addition, we will discuss its processing time and convergence of concentrations.

4.4.2.1 Validation of the policy selection

The policies in Section 4.4.1 are frequently used in operations of networks that provide services for power systems. All the outputs from our method are consistent with the judgment of the network operator. This reveals that our method is feasible for real operations.

However, outputs from our method are not always consistent with the operator's judgment, if a condition that is not written in the policies is considered. In addition, outputs from our method would not be feasible if policies do not reflect user requirements. We will mention this issue in Chapter 8 because an approach to solve this defect is relative to FDLU.

4.4.2.2 Processing time

The graph in Figure 39 shows a change in processing time until 1,000 steps of the calculation, for the number of policies. The order of calculations for the policy selection is relative to the square of the number of policies. This comes from the fact that the current program for our method deals with a full-meshed graph. This program considers all pairs of policies even if they do not have influence on each other.

There are from dozens to thousands of policies on each real operation. The latest workstations or PC servers perform at least twice the HP K250 that was used for the experiments. The concentrations of policies converge in 200 steps in almost all cases. In addition, the current program can be tuned up. These indications are that a set of policies can be selected in a couple of minutes, even if there are 1,000 policies. This performance is sufficient for the network operations such as task scheduling because these operations currently need a couple of hours.

4.4.2.3 Convergence of concentrations

We have gained the outputs in which all concentrations converged in the experiments. This comes from the following two factors. The first one is that the concentration of an antibody is apt to decrease because the affinity for suppression is stronger than the one for stimulation. The second factor is that conditions do not change during the calculations. The schedule to read conditions, therefore, is important for our method to be applied to real operations. However, we have not given theoretical proof of the convergence of concentrations. This is our future work.

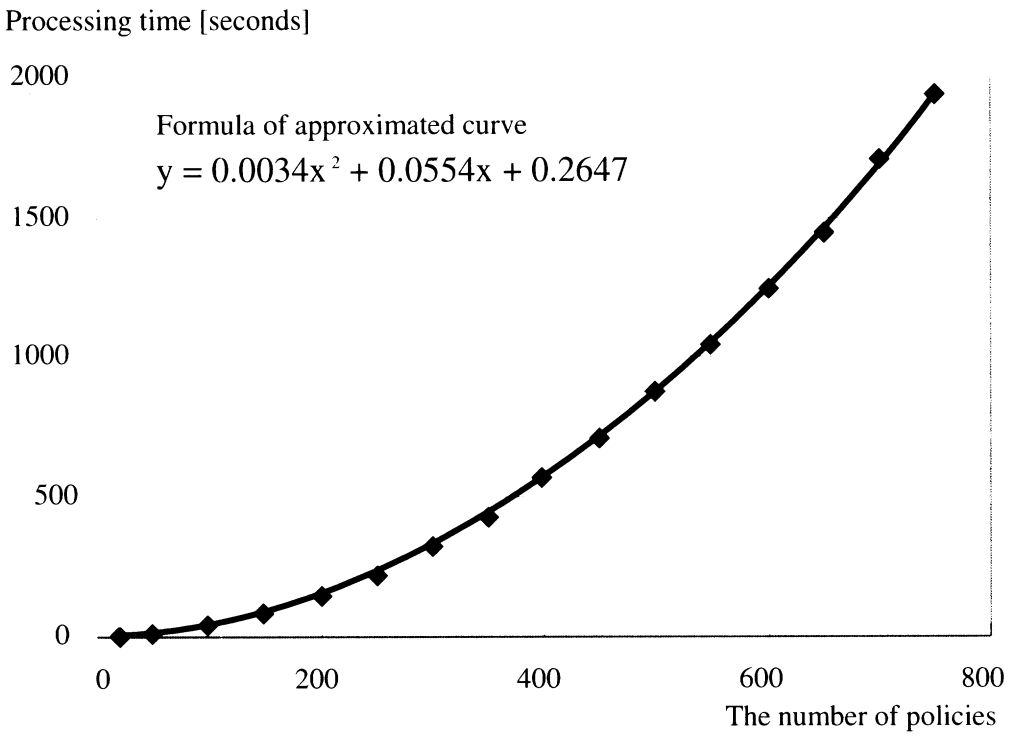


Figure 39: The number of policies and processing time

4.5 Summary of the policy selection

This chapter described the policy selection that makes an NMS adaptive to a change of policies. The method for policy selection makes use of the concept of artificial immune networks. We attached policies offered in the previous chapter to an element of the artificial immune networks. We changed the formula to calculate the concentrations of policies, and idiotopes that have different values for stimulation and suppression. The selection of multiple policies without conflicts has been achieved by these changes.

We have evaluated our method by application to network management to maintain the quality of various services. The results show that our contribution confers benefits on selecting policies, as mentioned below.

1) **Correct outputs**: The program based on our method selected one or some policies in the case study. These selected policies are validated to an actual operation of a

network. It is important that this program does not select a policy that must not be performed under a set of conditions. Many systems have not been accepted by real business because they produced outputs that were redundant or incorrect for a certain condition. Therefore, this feature makes the policy selection more applicable to real operations of networks.

2) **Practical calculation speed**: The program used in the case study finishes the selection in dozens of minutes, where it picks up policies from hundreds of policies. This means that our method helps jobs, e.g. task scheduling, that are not necessary to complete in a few minutes. There are a number of such jobs in actual operations. Therefore, our contribution improves the efficiency of such jobs. On the other hand, our method is insufficient for jobs in which a decision has to be made quickly. Instruction about a fault is one such job. System designers have to take account of jobs characteristics when this policy selection is applied.
3) **Simple parameter setting**: Outputs of the selection are influenced by parameters that are set relations between policies. If parameters are inefficient, an output may be meaningless for an actual operation even though the content of the policies is correct. Our contribution sets default values for each parameter, and rules to calculate values when multiple associations exist on a policy. This makes it simple to set each parameter and contributes to correct outputs.
4) **Adaptability to change of policies**: In our contribution, no formula needs to change when a policy is added, deleted, or changed. Parameters can be set simply according to the rules. These features make the policy selection accept changes of policies. A programmer or user has to define new formulas or make a system relearn the new policy set in other approaches such as neural networks or genetic algorithms. Policies are added, deleted, or changed many times in network operations so our contribution has an advantage to this field.

Chapter 5
Revising components

A function is invoked in accordance with a selected policy. If the designated function has been provided by an off-the-shelf software component, system designers can make use of it without change. If not, they have to customize functions provided in an off-the-shelf component or make a new component. This chapter proposes a method to change a function in an off-the-shelf component. This method offers four types of components for customizations. We have referred to them as "RevComponent".

"Without modifications of an off-the-shelf component" is the key of our proposal. Two types of risks arise from direct modifications of off-the-shelf components. The first risk is that a change may be costly, because source codes in most off-the-shelf components are usually closed to users or third parties. If users demand the supplier of the component to open the source codes, they may have to pay a large amount of royalty. The second risk is that users may not use upgraded version of an off-the-shelf component if it is modified directly. In this case, the modified off-the-shelf component provides the original function that is needed for user requirements. Its upgraded version, however, might not provide the original function. Users, therefore, cannot use the upgraded version without change.

These risks indicate that RevComponent must customize a function provided in an off-the-shelf component without change of the source codes of the component. In addition, RevComponent should provide several ways to customize a function, such as addition or deletion. Of course, system performance should not be degraded by introduction of RevComponent.

This chapter is structured as follows: first, related work comes to clarify existing approaches to customize functions. In the following section, we mention the details of RevComponents and the results of experiments with applications of RevComponents to an NMS. The next section puts forward the case for RevComponents characteristics, such as comparative performance. We summarize the proposal of RevComponent in the last section.

5.1 Related works

In this section, we show works related to the customization of functions. We review the work in terms of the customization after software is completed, or the adaptation of software.

Some recent systems are based on delegation. In these systems, a subsystem delegates tasks to other systems, by sending a script that describes procedures [113, 114]. These systems are able to fulfill individual requirements that arise on run-time, because the script is written at a stage distinct from analysis or design of the system. A script, however, has a tendency to be complicated due to scripting function as a whole. Suzuki et al. have simplified scripts using a system that sends a script describing only function flow, and delegated systems that build functions based on the received script [115]. Off-the-shelf components must have the mechanism that receives and executes a script if delegation is used in a component-based system. Users, however, cannot expect that all off-the-shelf components be built based on the delegation.

Mobile agents [116] can also modify apparently a function on hardware at run-time, since they carry executable codes. This technology seems to be promising for the redevelopment of systems. It has been applied to an NMS in several studies [117, 118]. However, its design and maintenance are still difficult when it is used with off-the-shelf components. In addition, off-the-shelf components are usually designed for common use to a number of network operators, so that mobile agents would not be adopted unless it has good features in terms of performance or design.

Components themselves have become more adaptive in recent researches. Java™ Management Extensions (JMX) [119] define a framework for dynamic MBeans that are components for system management in a Java community. The dynamic MBeans can change their exposed methods and behaviors after design of components. A system obtains information about the methods via an interface that is defined in JMX. In this framework, a change in an off-the-shelf component may require another component to change, too. Studies on Adaptive Plug-and Play Components (APPC) [120], and Active Q Adaptor [121], also proposed components that can change their functions. These studies, however, do not show a methodology that changes functions without a change in off-the-shelf components.

Wegdam et al. [122] proposed a system that makes use of interceptors as well as RevComponents. It focuses on management of the CORBA platform, while RevComponent is used for management functions. Therefore, it does not take into account how to change a function provided by an off-the-shelf component.

RevComponent, which is proposed in this section, has a role as a software wrapper. Software wrappers have been applied for various domains, e.g. security [123, 124, 125], integrating API (Application Programming Interface) [126], or manipulation of XML [127, 128]. Some of them add new processing for their purposes and some of them have offered code

generators to facilitate building wrappers. However, they take no account of how to modify processing under the environment based on distributed components. Therefore, it is not clarified that a software wrapper can modify a function provided by another software with its maintainability and lightweight processing.

5.2 Proposed technique

We set out our proposal for RevComponents in this section. First, we mention the requirements for RevComponents to change functions to fulfill user requirements, with no change of off-the-shelf components. Following to the requirements, we draw the positions and roles of RevComponents in a run-time environment. We also show the configuration of RevComponent and its templates to simplify its construction.

5.2.1 Requirements for designing RevComponents

In this subsection, we clarify six requirements for RevComponents.

1) **Ability to modify functions**: Various changes of functions may occur, for technical or non-technical reasons. It is hard to predict details of possible changes before the requirement actually arises, as mentioned in Section 2.2.5. Accordingly, it must be possible to modify or remove a function provided by a component that has been used. In addition, it must be possible to add a new function not provided by the component, and to cope with sending messages to a new interface not employed so far.
2) **Simple customization**: Off-the-shelf components should be used to construct an NMS, and are distributed to a number of machines. This type of construction needs knowledge about distributed systems. The same knowledge is required to construct a RevComponent because it behaves in the same way as other components in an NMS. However, a designer of RevComponents may not be expected to have complete knowledge, as they might be third part vendors that have not constructed such an NMS before. Therefore, it is useful to provide a framework to help make RevComponents without full, detailed knowledge about distributed systems.
3) **Preventing the slowing-down of processing**: An NMS performs some functions that are time-sensitive, such as alarm diagnosis. The processing of these functions should not be delayed by the introduction of RevComponents. RevComponents should perform processing for function change, in the same or less time than other methodologies. Accordingly, RevComponents should include mechanisms that

prevent processing time from being prolonged.

4) **Maintaining small size**: A function may be modified repetitively after an operator starts to use an NMS. A new service, or business process reengineering, can cause a change of functions. RevComponents should keep small volume of memory for its execution, even though it is used several times for the modifications. This supports maintenance, keeps its processing time short, and/or prevents the using up of processing resources.
5) **Easy plug-in/removal**: It is desirable to be able to modify a function partly, because various workers make use of an NMS for their own operations. If RevComponents can be plugged in and removed from the NMS, it can quickly fulfill an individual's requirements.
6) **Appropriate testing**: One important condition is to confirm how a RevComponent performs. Tests of RevComponents must show if it fulfills requirements. Tests must also reveal any side effects on other components. These tests should be executed in a practical environment, although they must not have an effect on operations and processes. Therefore, a sensible and effective means of testing RevComponents is required.

5.2.2 Roles at run-time

A RevComponent is placed before or after the component whose function is to be modified. The component that the RevComponent is related with is referred to the target component in this section. When placed before a target component, a RevComponent intercepts requests sent to the target. When placed after a target component, a RevComponent receives requests sent by the target. In this thesis, we refer to a RevComponent before a target component as Precedent RevComponent, and one after a target as Subsequent RevComponent.

5.2.2.1 Role of precedent RevComponent

Precedent RevComponent takes a specified request in advance of its target in order to modify a function. It has the same interface definitions as those of the target. It processes data and/or sends other requests based on the received request in order to satisfy an individual requirement as illustrated in Figure 40. The RevComponent forwards the same request to the target component in cases where functions provided by the target are used.

A Precedent RevComponent can be designed not to forward the request to the target component as shown in Figure 41. In this case, the functions performed by the target component are left out, since the target does not receive the request. We have called the first type preprocessing, and the second type alternative processing.

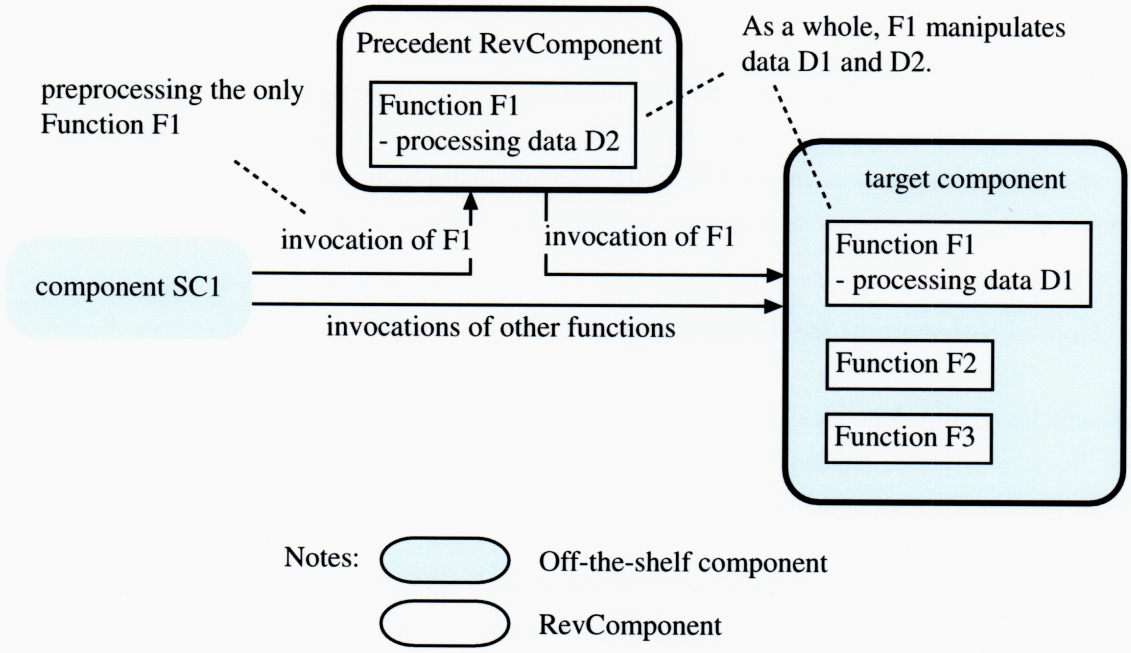


Figure 40: Precedent RevComponent for preprocessing

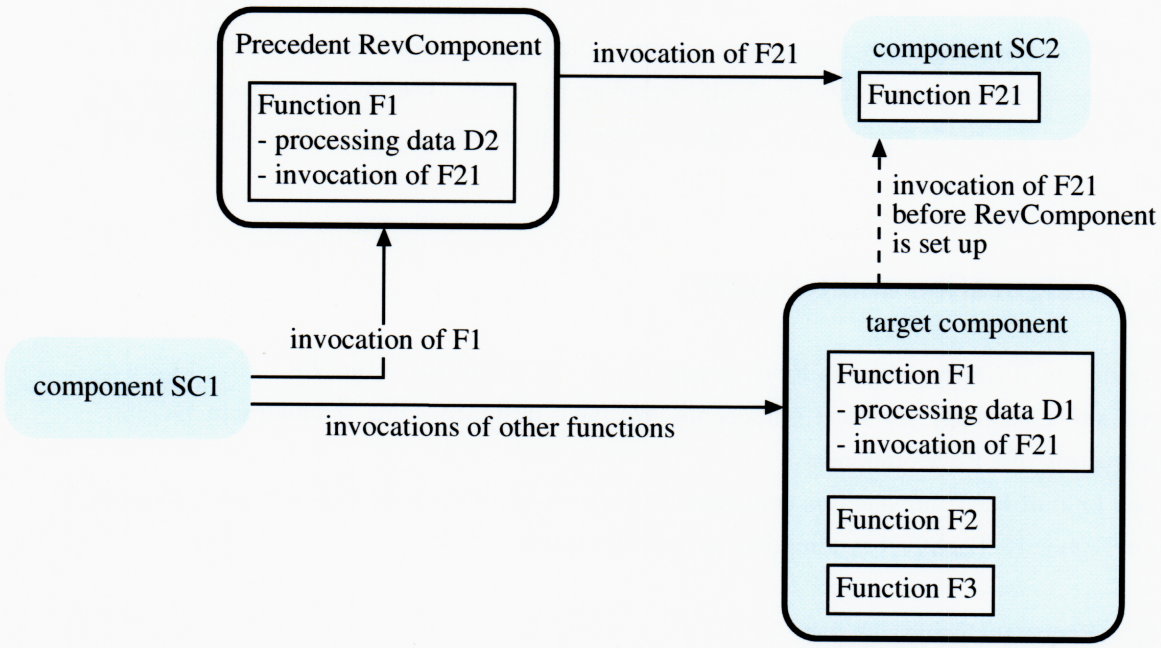


Figure 41: Precedent RevComponent for alternative processing

New data and/or procedures can be added to a function provided by a target component, using preprocessing. A new calculation of statistics, for example, can be added for a new service deployment.

Alternative processing skips the function invoked by the request that the Precedent RevComponent receives in advance of the target. This provides the means to deactivate an old function, and activate a new one, in order to fulfill an individual requirement. Other functions, except for the deactivated ones, are provided by the target component as before.

5.2.2.2 Role of Subsequent RevComponent

A Subsequent RevComponent picks up requests sent by its target component and carries out functions based on these requests. Each Subsequence RevComponent defines the same interfaces as those of the components to which the target sends the requests. These functions can be categorized into two types.

The first type transforms interface syntax and semantics when a new component with new interfaces is introduced into an NMS as illustrated in Figure 42. We refer to this function as interface changing in this section. It allows components that have been used to cope with a change of environment.

In the second type of function, information picked up by a RevComponent is distributed to other components as shown in Figure 43. We refer to the second type as information distribution. The target component is not concerned with the number of components that receive the request. In addition, routes along with the request is transmitted can be selected and altered by this function after operators start to use the NMS including the target. The target component can be adapted to the modified organizations and/or business processes to be supported using this function.

5.2.3 Configuration and templates

We have designed configuration in RevComponents on a CORBA platform. We selected C++ as a programming language for RevComponents. We have designed some templates to facilitate the construction of a RevComponent.

Any kind of RevComponent is composed of some or all of the following four parts that are shown in Figure 44. Each part is composed of several objects.

1) Request transmission part
2) Processing scenario part
3) Internal data processing part
4) Request preparation part.

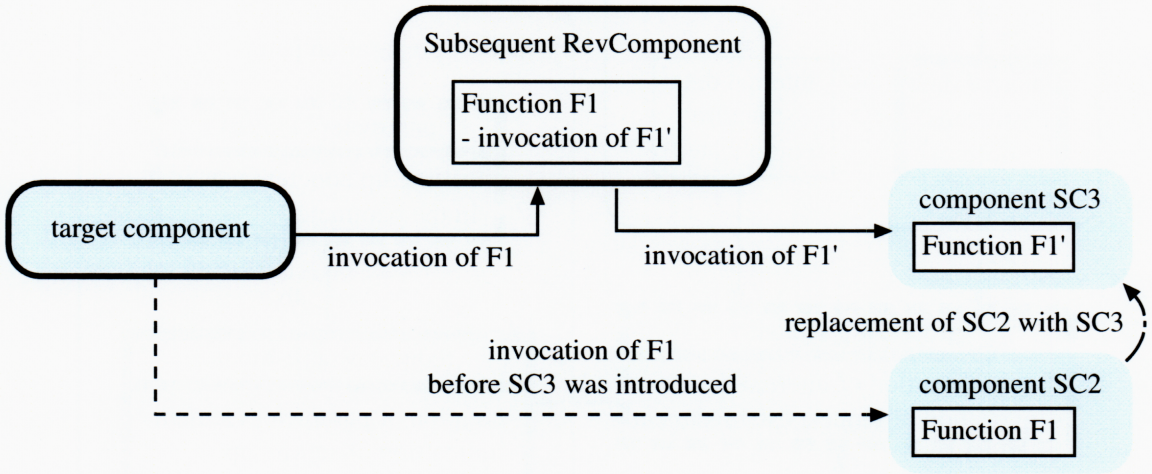


Figure 42: Subsequence RevComponent for interface changing

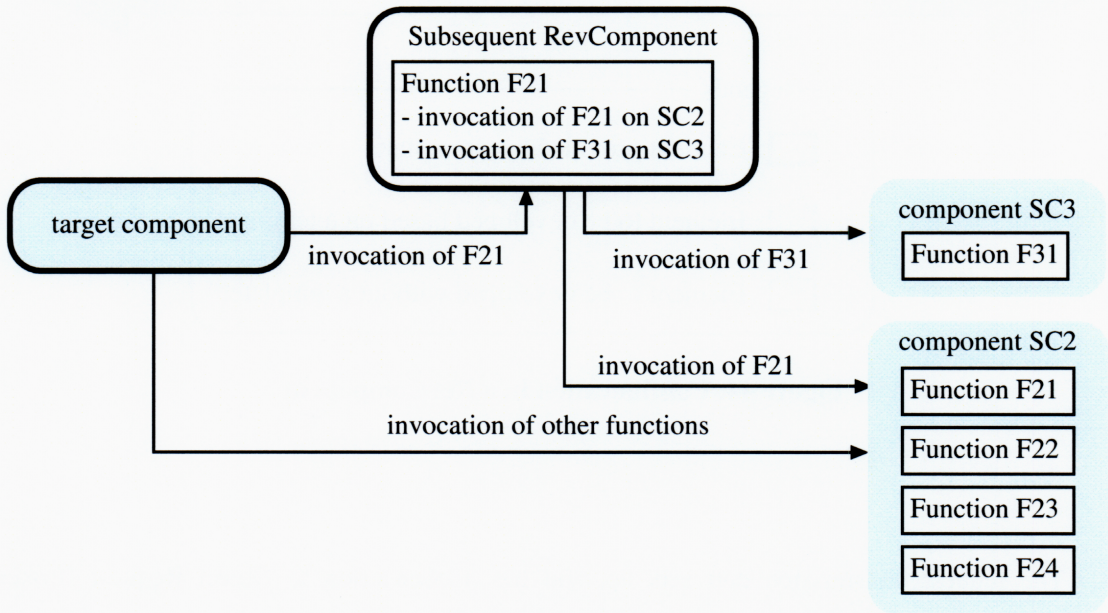


Figure 43: Subsequence RevComponent for information distribution

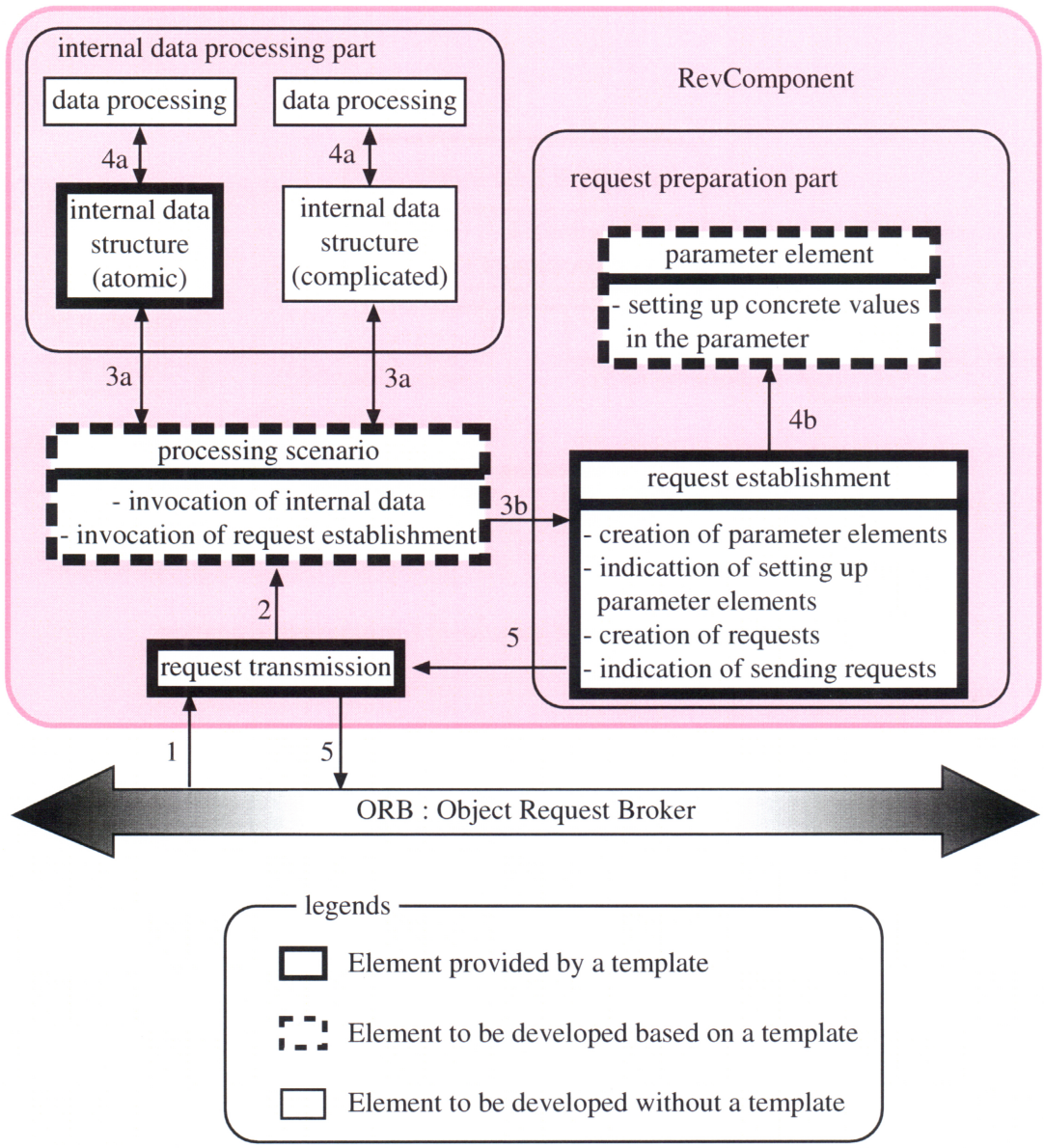


Figure 44: Configuration in a RevComponent

The request transmission part sets up, and has a connection to Object Request Broker (ORB), which conveys requests between components. The template for this part provides the creation of an object that implements the interface, initial value loading, and start of a routine for receiving messages.

The processing scenario part invokes methods in the internal data processing part and/or the request preparation part. A framework is provided in the template for these invocations. The

template is derived from the interface definitions of its target component or components that receive messages from this RevComponent. It is necessary to decide that methods are called in this framework, in order to realize a function for an individual requirement. The processing scenario part forwards the same request picked up by the RevComponent to the components that are original receivers of the request. This request forwarding is used in the case of the preprocessing and the information distribution. The request is forwarded by source codes written in the template for this part.

Data used in the RevComponent is manipulated in the internal data processing part for a new function. Although templates are provided for atomic data structures such as strings or integers, the means of setting the values must be designed individually. No template is prepared for complicated data composed of several data. This is because the data structure is designed according to each requirement. However, programmers can use libraries in which many data types are defined as C++ classes.

The request preparation part makes a request that the RevComponent sends for its own functions. This part is used in RevComponents for the following three types: the alternative processing, the interface changing, and the information distribution. This part makes use of interfaces defined for components other than the target. A template provides the client stub of procedure, since it can be derived from the interface definition of the request to be made. Each element in a parameter of the request is implemented based on a template that provides procedures for setting up the values. Specific procedures for setting up the values need to be developed.

Typical procedure of a RevComponent is described as follows. The numbers are corresponding to them in Figure 44.

1) Receipt of a request
2) Invocation of the processing scenario
3) Delegation of tasks to the internal data processing part and/or the request preparation part
4) Process of internal data and/or indication of parameter set and response
5) Transition of a request

5.2.4 Code generator

We have offered a code generator that produces source code and header files to be compiled to a RevComponent. Figure 45 illustrates input and output files that the code generator reads and writes in the case where a RevComponent compiled from C++ runs on CORBA. CORBA is the most popular middleware of distributed object environment for NMSs. C++ is the most popular programming language for CORBA as well as Java™. The code generator per se is written in Perl.

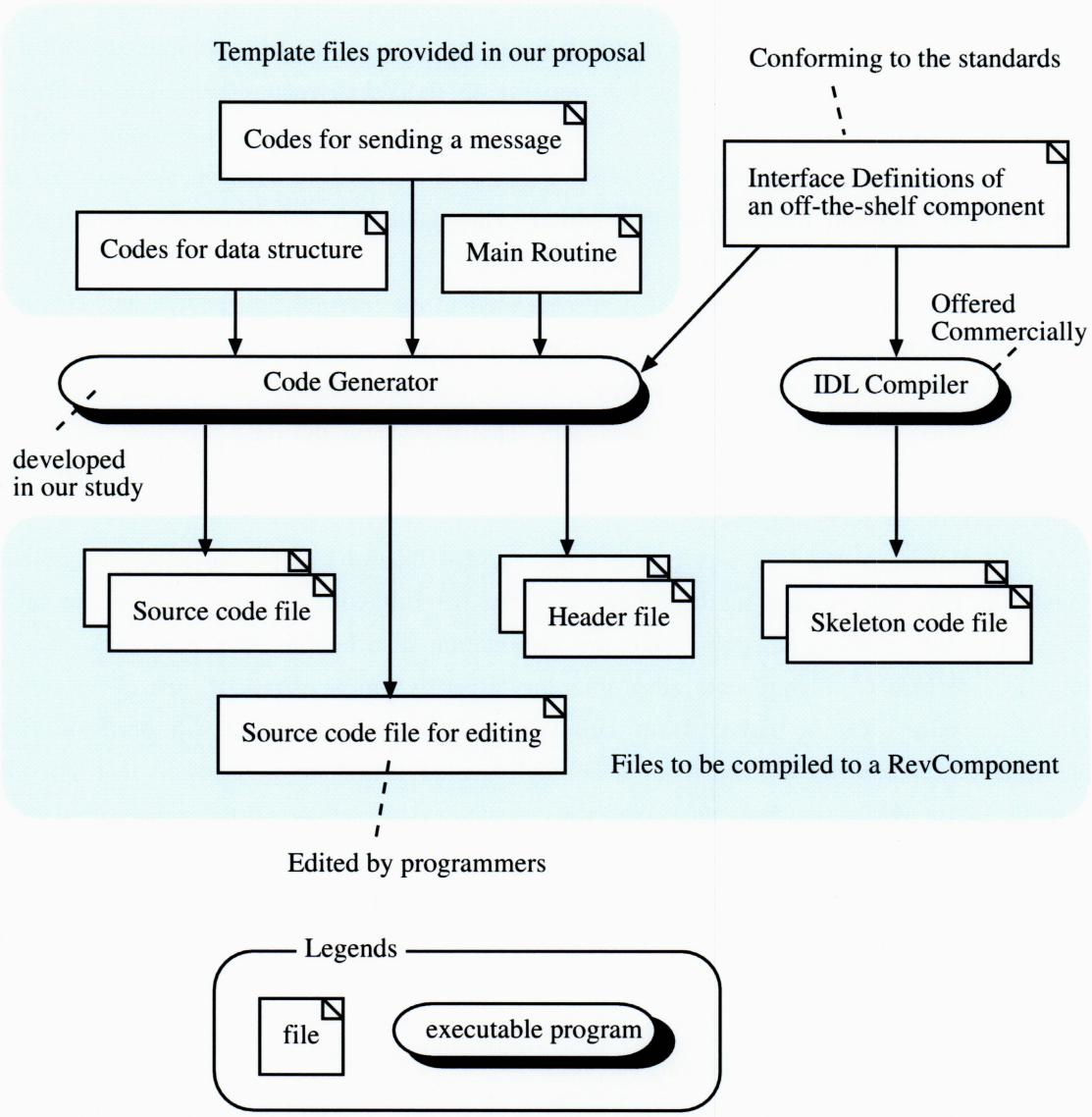


Figure 45: Code generator for RevComponents on C++ and CORBA

The code generator read four files that have been provided in advance. The file "codes for data structure" contains templates for internal data structure (atomic) mentioned in the previous section. The file "codes for sending a message" contains templates in the request preparation part in Figure 44. The main routine file includes the template of the request transmission and codes for initializing a RevComponent. In addition, it has the template of the processing scenario so that its codes send the picked up message to the target component. The file "Interface Definitions of an off-the-shelf Component" contains interface definitions of the target

or another component that will receive messages from the RevComponent. These components are off-the-shelf so that the definitions are provided as a standard.

There are two executable programs to produce files to be compiled. One is an IDL compiler that is usually provided commercially. Some of them can be used without charges. It reads interface definitions of an off-the-shelf component and produces skeleton files. The skeleton files provide a RevComponent with mechanisms of a client stub or server skeleton in CORBA.

The other is the code generator. It produces three types of files to be compiled to a RevComponent; source code files, header files, and a source code file for editing. These files are composed of templates and codes generated from the interface definitions. Methods that have to be customized for each RevComponent are stored in the source code file for editing. Other files are not needed to change for each RevComponent.

5.3 Evaluations using a prototype system

We have built a system to evaluate processing time and customization with RevComponents. This system is composed of four types of components equivalent to off-the-shelf ones, and a Precedent RevComponent, as illustrated in Figure 46. A SQM (Service Quality Management) component, which is regarded to be off-the-shelf, is a target in this system. The SQM component and a RevComponent are installed onto the same workstation so that the messages between them do not pass through a network. Other components send a message to on another via LAN. There are fore EM (Element Management) components. They emit messages to the NDM (Network Data Management) component at 5 seconds intervals.

Each message in this system needs its own response from a server component, which receives the message. Accordingly, a client component, which sends the message, has to wait for the response and cannot do anything in a thread until the response comes.

This RevComponent performs some or all of three types of preprocessing when it receives a "Network Usage/Performance Trends" message from the NDM component. The types of preprocessing are as follows:

Type I: Increasing a counter by one
Type II: Creating an instance of a standard object for logging and saving the data on a file
Type III: Calculating difference between inbound and outbound communications

In the type III process, the RevComponent sends a message to the NDM component in order to gain information about delay inbound/outbound communications. In other types, it does not send any messages to another component.

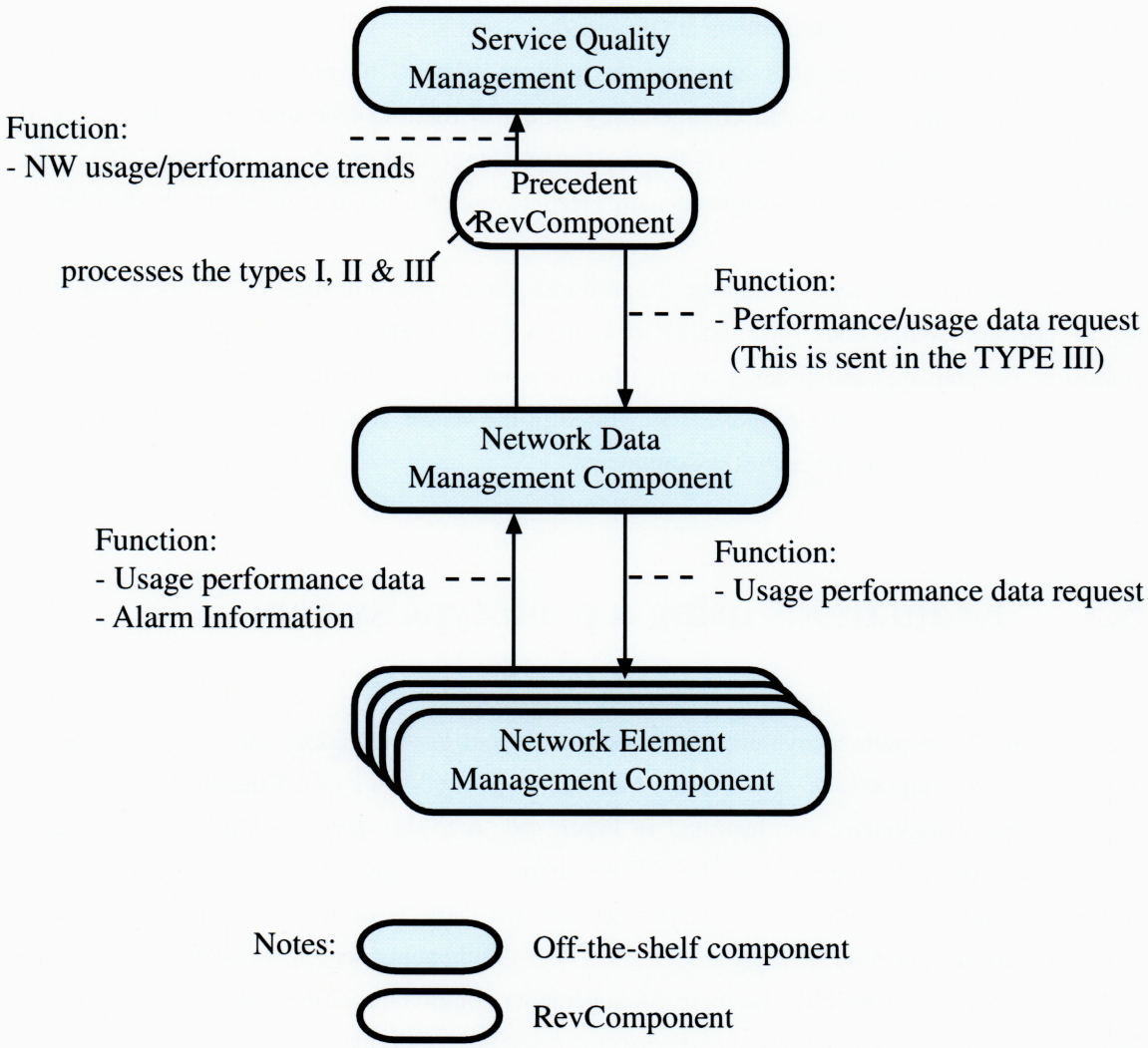


Figure 46: An evaluation system for RevComponent

5.3.1 Performance

We measured system performance and show its results in Figure 47. Each processing time shows the average value of 100 executions of each type. For each type, we measured the system performance in three cases. A modified SQM component performed the specified types of preprocessing in the first case. The modified SQM is corresponding to an off-the-shelf component that is modified for fulfilling user requirements. The results of this case are shown

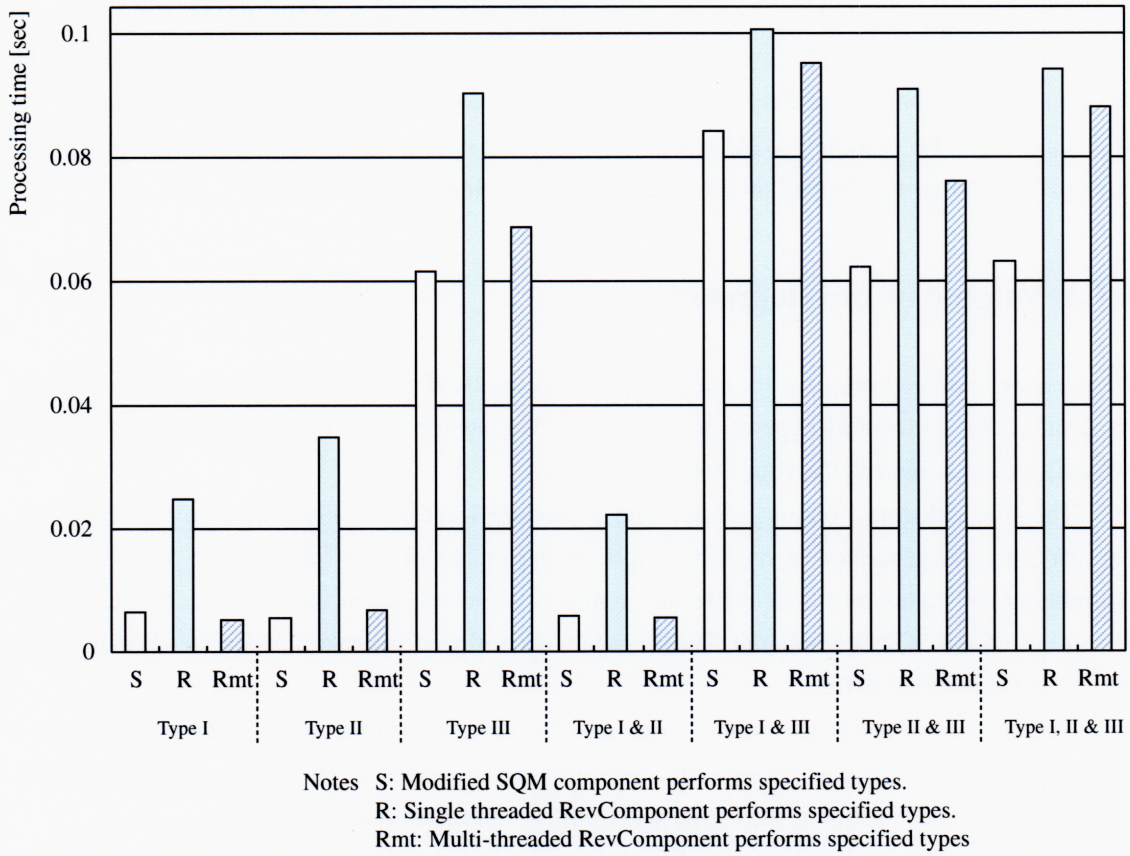


Figure 47: Performance by preprocessing types and performing components

with an "S" in Figure 47. A single-threaded RevComponent performed the specified types in the second case. It sends a message to the SQM component in the evaluation system after processing the specified types. The results of this case are shown with an "R" in Figure 47. A multi-threaded RevComponent performed the specified types in the last case. It sends a message to the SQM component and processes the specified types of preprocessing in parallel. The results of this case depicted with "Rmt" in Figure 47.

The processing time of the single-threaded RevComponent is at most about 30 mille-seconds longer than that of the modified SQM. The difference in performance does not depend on the type of preprocessing.

On the other hand, the processing time of the multi-threaded RevComponent is shorter than that of the single-threaded RevComponent. In addition, it is equivalent to the processing time of the modified SQM when preprocessing of the multi-threaded RevComponent does not include the type III. Even if the type III is included, its processing time is at most about 20 mille-seconds longer than that of the modified SQM.

These indicate that the difference between the performance of the RevComponents and that of the modified SQM comes from sending a message to the SQM component. The RevComponents, that is, receive a message from the NDM and send the same message to the SQM, although the modified SQM does not send a message to execute the original processes. The difference does not depend on the type of preprocessing.

5.3.2 Volume of Development

We measured source codes for the preprocessing in the RevComponents and the modified SQM component that perform all type of preprocessing. We put the codes of the RevComponents into two categories to find out the volume of development.

The first category contains source codes produced automatically based on the templates. We made a code generator to support simple programming of RevComponents. This generator assembles C++ codes into a template and replaces some tokens in the template based on IDL definitions of the SQM component. Programmers do not need to make these codes because they are generated automatically.

Source codes of a programmer's own making are put into the second category. To put it concretely, they are for calculating data, and emitting messages in each type of preprocessing. This volume of codes actually shows the workload of a programmer.

Figure 48 shows a LOC (Lines of Codes) of the RevComponents, and that of the modified SQM component. This indicates that workload for building RevComponents is almost the same as that for modifying the SQM component. In addition, the multi-threaded RevComponent can be built without codes in addition to those of the single-threaded one. This is because the generator produces codes to make a RevComponent multi-threaded.

5.3.3 Plugging-in and Removal of RevComponents

The RevComponents can be plugged in and removed from the evaluation system by setting object implementation names for the RevComponents and its target component. These do not affect source codes of other off-the-shelf components in the system. No difference apart from the names is seen from the viewpoint of other off-the-shelf components. The difference in names does not matter because object implementation names can usually be set as a parameter in off-the-shelf components.

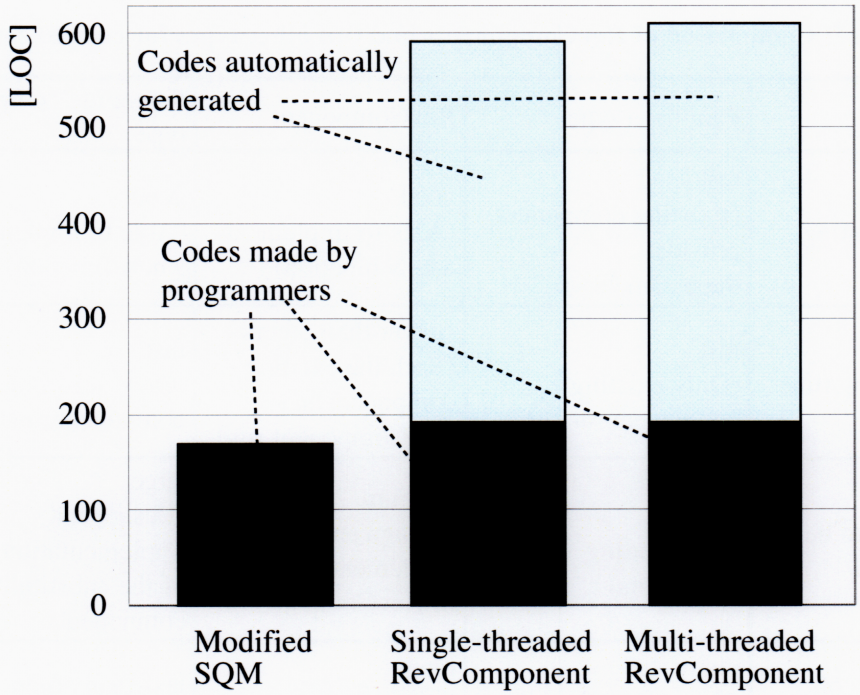


Figure 48: Volume of source codes in components for the preprocessing type I, II, and III

5.4 Considerations

In this section, we discuss the characteristics of RevComponents through comparison with other methods of customization. We classify these methods into two types for ease of comparison with RevComponents. The first includes changing the behavior of a component via its parameters. The second is when a component is replaced with another providing customized functions. The modified SQM component in the evaluation system is put into this type. In this section, we refer to the first type as the parameter type, and the second as the substitute component type.

Table 7 shows comparisons between these two types and RevComponents. In the following subsections, we consider RevComponents based on these comparisons.

Table 7: Comparison of RevComponents and two other types for customization

	Parameter type	RevComponent	Substitute component type
Ability implementing functions	restricted (Existing component provides the parameters)	good (Able to implement a new function)	good (Able to implement a new function)
Processing time	fast (Only existing component runs)	slower than others (Both the existing component and RevComponent run)	fast (Only substitute component runs)
Volume to be developed	small (Only setting parameters)	medium (Design and implementation of RevComponent)	large (Design and implementation of the substitute component)
Diversion of existing components	many (Diversion of existing components without redesign)	many (Diversion of existing components without redesign)	less than others (Abandon existing components)
Adaptability to component upgrades for general purposes	easy (Setting up corresponding to new version)	easy (Plug-in/remove new version)	more difficult (Cannot use the new version without modifications)

5.4.1 Compatibility between adaptability and reusability

The parameter type may not be adequate under the circumstances to modify functions, because it is restricted to a range predicted at the design stage of the component. The substitute component type can provide new functions and data that satisfy individual needs. In addition, it can change its interfaces that are exposed to other components. However, it may not utilize the merits of an off-the-shelf component that has many functions able to be used without modification. On the other hand, customization with a RevComponent is superior to the other two types. It can provide a new function while reusing some functions provided by off-the-shelf components that are adequate for requirements.

5.4.2 Workload for development

The generator based on the templates produces the codes needed for a RevComponent described in Section 5.3. This reduces the amount of RevComponent development work to almost the same as that for new functions in a substitute component. However, we cannot use the generator if interface definitions have to be changed in order to fulfill a requirement. In such cases, programmers may have to develop more codes for a RevComponent than for a substitute component. Accordingly, the key point for development of RevComponents is whether interface definitions are changed or not.

The workload to develop RevComponents is also reduced by easy its scheme of the plugging-in and removal. Off-the-shelf components are preserved in their original state when a RevComponent is installed into the NMS. This avoids unnecessary work due to component damage. It also makes it unnecessary to understand the contents of off-the-shelf components, as components can be reused as a black box in this framework. These are strong arguments against using the substitute component type.

Workload for the parameter type is less than that for RevComponents, since it is not necessary to develop new source codes. It is unadvisable to use a RevComponent in cases where a component can adapt to a requirement by setting parameters.

5.4.3 Performance

The experiment described above indicates that a RevComponent may not cause delays in processing in the case of minor changes. However, a RevComponent is inferior to the other twp types in terms of performance. This is because the target component performance follows a RevComponent, or vice versa. If a function is sensitive to time, it may be better to customize the function using the substitute component type.

5.5 Summary of RevComponents

We are proposing the use of components known as RevComponents. These components eliminate the need to replace an off-the-shelf component with new one that provides new functions. They are located before/after a target component. They modify or remove an existing function, or add a new one. The roles of RevComponents are as follows: preprocessing for additional functions and/or data, alternative processing for the deactivation of existing functions, interface changing, and information distribution.

We have described an evaluation system for RevComponents applied to service quality

management. The results of these experiments show that RevComponents confer benefits on a change of functions and black-box reuse of off-the-shelf components, as mentioned below.

1) **Various customizations**: A system designer is able to add, delete or change a function using only RevComponents. In addition, RevComponents do not have any impact on the standard scheme. This helps an NMS to be composed of off-the-shelf components and components specific to a network operator.
2) **Simple construction**: Our contribution provides templates to facilitate the building of RevComponents. Some of these templates are derived from the interface definitions of a target or related component. They allow developers to concentrate on the development of new functions, since they encapsulate the details of data structure and message forwarding to the target component. The volume of coding for a RevComponent is the same as that for modifying an existing component. The codes for a RevComponent are separated from the codes to implement functions conforming to the standards for network operations. Using our contribution, an added or modified function can be maintained simply in the field of network management.
3) **Simple installation and removal**: RevComponents can be plugged in and removed without complicated configurations. This feature enables a function provided by a RevComponent to be tested. This is important for an NMS because a combination of a RevComponent and an off-the-shelf component cannot be verified except for tests on the NMS. This also facilitates the maintenance of a function that is added or changed.
4) **Lightweight processing**: Processing time did not get much longer when a multi-threaded RevComponent was used in a case study. In particular, the processing was not delayed in the case where a RevComponent did not interact with a component except for the target. This shows that a RevComponent can work well in an actual operation of networks.

Based on these results, we compared the characteristics of RevComponents with other methods. We can conclude that a RevComponent is suitable for cases where a requirement cannot be satisfied by tuning parameters in a component, and the modification does not affect interface definitions.

Chapter 6
Partial Extension Package

Functions must be developed from design level in the area that RevComponents cannot perform well. In this case, it is better to build a function based on the standard objects than from scratch. The standard objects are solid so that the usage of them can contribute to shortening of function developments.

We focus on the extension of a function in this section because functions offered by the objects can be reduced easily. We propose a partial extension package for the design of object classes that provide a custom function. This package is attached to the standard objects to add new information processing. It is composed of several design patterns [129] to eliminate the influence on the standard objects and to maintain reusability of the classes.

This chapter is structured as follows. Section 6.1 summarizes work that make software reusable in cases where a function is customized at design level. Section 6.2 defines flexibility with respect to the partial extension package. In Section 6.3, we set out our proposal for the partial extension package, and describe its applications to network operating processes in Section 6.4. In Section 6.5, we measure the applications in terms of the flexibility. Section 6.6 concludes the chapter.

6.1 Related works

Object-oriented technology is most popular and powerful to analyze, design and implement a computer system. Key concept of this technology includes inheritance, encapsulation, delegation, and so on. These features allow designers or programmers to make and maintain easily their artifacts. Many methods have been proposed to elicit advantages of the technology [130]. Recently, RUP [6] is a dominant method because it integrates Booch Method [131], OMT [132], and OOSE [1], which were major methods before RUP. These methods bring up making classes from scratch in terms of design.

Design patterns are also an important technology for class design [129]. It offers a number of patterns to make class configuration flexible and reusable. Each pattern is composed of a few or several classes. It has its own purposes and conditions that are suitable to use it. For example,

strategy pattern defines a set of algorithms, and encapsulates each one in a class. These classes can be exchanged independent of a client class. This pattern fits to conditions that only behavior of relevant classes is different and other features are same. When they are used, designers combine them to built a system. It depends on designer's skill to select which pattern is used. A research has revealed that the design patterns are beneficial by empirical evidence [133].

A framework is a set of classes that make up a reusable design product for a certain application. For example, a framework is provided for the specification of distributed systems [134]. Another example is a framework that assists to build a financial system [135]. System architecture is defined in a framework. It sets how to assign responsibility to objects, how an object interacts with others, etc. Main programs in a framework are reused when a designer builds a system. Frameworks are different from design patterns although they have several similar features. The major points are as follows.

1) Design patterns are more abstract. Frameworks are more concrete.
2) Design patterns are smaller elements. Frameworks are larger elements.
3) Design patterns are more general. Frameworks are more specific to certain domains.

These technologies seem to be tough to extend standard objects in regard to functions, while the objects are kept reusable. There are too many options for designers to extend the objects in case where the object-oriented technology is used without design patterns or frameworks. Design patterns are general and small elements so that a little larger grain of elements is useful to the extension. Frameworks can be applied to whole architecture of an NMS. Designers, however, cannot change or extend class structure or interactions so that it is difficult to extend the functions using them. Therefore, we need a fine-grained package to extend the standard objects.

6.2 Flexibility in function change

In this section, we define "flexibility" in terms of changing the behavior of a function in an information model composed of objects. This clarifies how the PEP is attached to the standard objects in a flexible manner.

An information model should have one or both of the following characteristics for changing the behavior of a function in a flexible manner. First, an information model can provide suitable functions with no modifications in the case of a change in requirements. We refer to this type as an adaptable information model. Second, a new object for a new function can be added with no, or only small, modifications of other parts in the information model. We refer to this type as a re-configurable information model.

Setting a parameter in the adaptable information model can change the behavior of a function. This does not require modification of the model. It is not sufficient in cases where a function has to be modified on a certain scale, although it does provide an easy way to change behavior.

The re-configurable information model provides flexible mechanisms that create and modify object classes to change functions on a large scale. However, it is difficult to design such models, because the characteristics of a model depend on an application, or the context to which the model applies. A designer of object classes sometimes comes up against the question of which physical or logical unit should be modeled as an object class.

The necessity to change a function depends on the policies of individual operators. Policies are affected by external primary factors e.g. technological progress or new services. These factors make it impossible to forecast that requirements might arise in the future, as mentioned in Section 2.2.5. Accordingly, we focus on the re-configurable information model that can realize a large-scale change in a function.

In particular, the model should have the following two properties in order to change behavior in a flexible manner.

1) Reusability of object classes: Object classes that have already been used in the case of function changes should be reusable with little or no modifications. These objects are referred to as existing objects.
2) Localization of modification: Object classes in a model should not be affected by classes that are added to the model for new processing and/or data. This additional object is referred to as a custom object.

Any custom object class can be easily added to, or removed from, existing objects, by the first property. Work to design custom object classes is reduced by the second property.

Both the reusability and localization in a model are relevant to a concept called 'coupling' between object classes. Stevens et al. first introduced the coupling concept in the context of structured development. They defined coupling as "the measure of the strength of association established by a connection from one module to another" [136]. This concept has been migrated to object-oriented design by Coad and Yourdon [137]. It has been reported that weak coupling makes it easy to reuse objects [138], and localizes the influence of modification of an object class [139]. Accordingly, it is most important to weaken coupling between an existing object and a custom object or between custom objects themselves.

6.3 Partial Extension Package

In this section, we set out our proposal for a partial extension package that changes behavior in a function in a flexible manner. This package is composed of custom objects, and is attached to existing objects to make them part of an information model for constructing an NMS. The proposed package will be described after the requirements for the package.

6.3.1 Requirements for the package

The PEP needs the two properties described in the previous section. The package must have the following requisites to achieve the above.

In the first place, information processing needs to be added, modified, and/or deleted in most function changes. These changes in information processing are realized in a custom object. This object invokes methods provided by other objects, and/or calculates values on its attributes. Coupling between a custom object class and others, especially existing object classes, must be kept weak in an information model using the package. The package should have features for the weak coupling to maintain the flexibility.

In the second place, states invoking information processing may by required in the function changes. Therefore, developers may need to design new states that have not been defined in existing objects. In contract, most substances related to the states (e.g. network elements or services) have been defined as standard objects by international organizations. The package needs to provide a facility that weakens coupling a custom object showing a new state and an existing object describing a substance.

In the third place, developers cannot really foresee which method, and/or data, will be required by future information processing. This makes it impossible to provide all the methods, and/or data, in an object manipulated by a custom object in advance. If an existing object were modified directly, it would force other object in the information model to change. Therefore, the package should provide a way to add new methods, and/or data, with no modification of existing objects.

To sum up, the package should provide features with the following characteristics, for changing a function in a flexible manner.

1) Ease of extension/modification/deletion of information processing in a custom object
2) Ease of setting a state invoking information processing
3) Avoidance of direct modification of existing objects for new information processing

6.3.2 Structure of the package

Figure 49 shows the structure of the PEP. This figure is a class diagram in UML, and is used to describe object classes with attributes, methods, and relations. A gray boxes in this figure stands for an object class for general-purpose use (an existing object). A white box in the figure indicates a custom object for changing a function in a flexible manner.

The PEP is composed of three object groups, as follows.

1) An object group showing a state invoking processing
2) An object group processing information
3) An object group providing additional data

The requirements described in the previous section, are fulfilled by these object groups. Each object group makes use of several design patterns to be reusable and weaken the coupling.

The structure and characteristics of each object group will be described in the following subsections.

6.3.2.1 Object group for states

This object group includes a state aggregation class and several state classes. Concrete state classes are derived from a state class.

A state aggregation class is related to a context object class, which is an existing object for general-purpose use. An instance of this class is created corresponding to an instance of the related context class. It also has an association to state objects, and forms a bridge between these two types of objects. This class provides three methods for state transition and invocation of information processing. One type of method related to state transition, e.g. method1() in Figure 49, is invoked in a context class. Another method related to state transition is changeState() in Figure 49. This method is called by a concrete state class for changing the relationship between the state aggregation class and concrete states. This invocation makes the state aggregation class move from the current state to the next. The method related to invocation of information processing is called by a context class. It contains some pointers to objects that are manipulated by the called information processing. It invokes methods provided by a related concrete state class, and sends the pointers to the class.

A state class describes an individual state relating to the context class. For example, a state class is designed in accordance with the operational state of a network element. A state class provides method interfaces for state transitions and/or invocations of processing. These methods do not include concrete behavior and an instance of this class is not created.

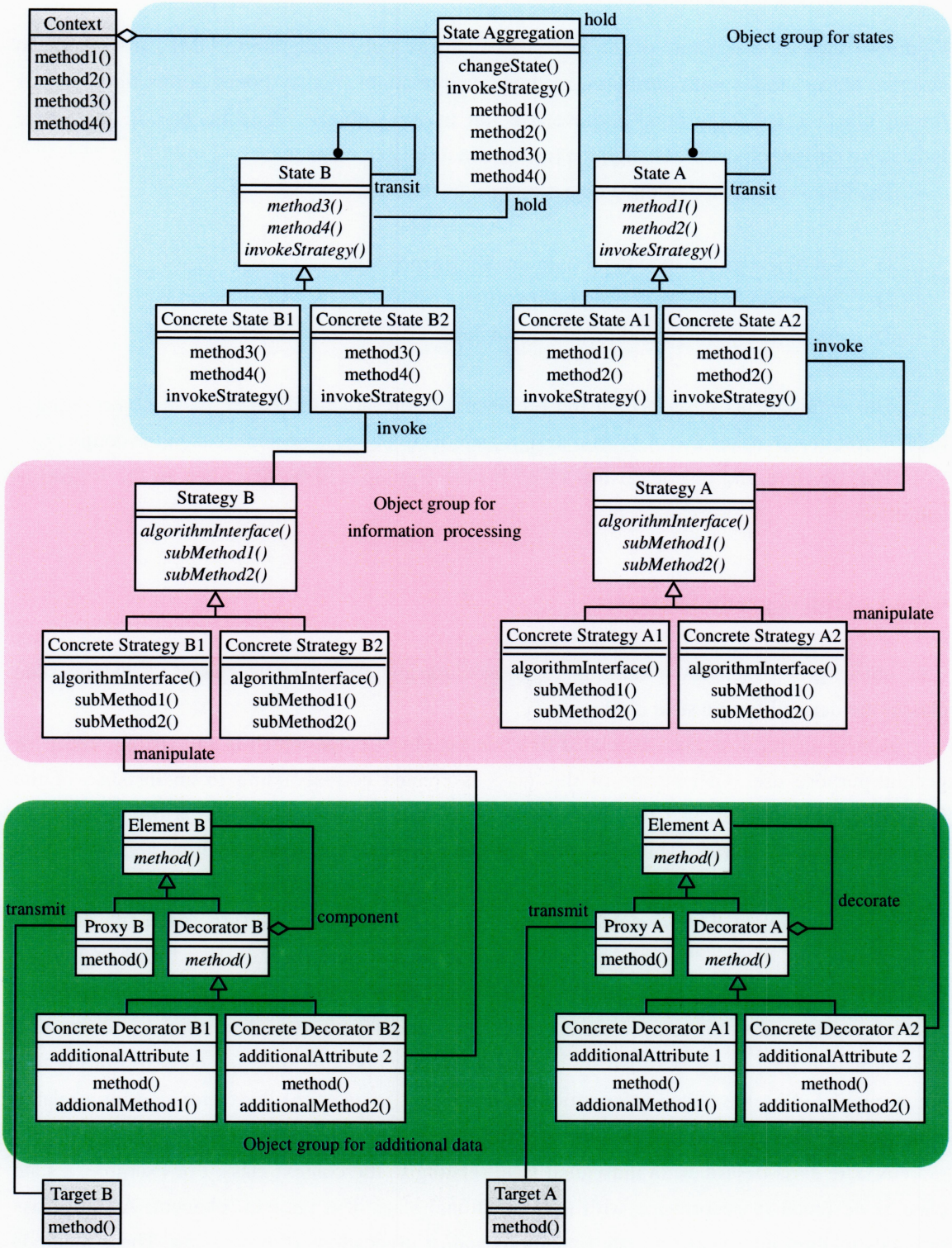


Figure 49: A class diagram of the Partial Extension Package

A concrete state class provides only the behavior of methods defined in the state class from which it is derived. This behavior calls information processing and state transition. A concrete state class has relations to other concrete state classes for state transition. It also has a relation to a strategy class that provides appropriate processing. An instance of this class may be created by only one in a system and shared by instances of the state aggregation class.

This group eliminates the influence on a context class, as to definitions and transitions of states. The state aggregation class deals with all relationships between a context class and state classes, reducing modifications of a context class. The state transition by concrete state classes does not require modifications of a context class.

6.3.2.2 Object group for information processing

This object group includes a strategy class and concrete strategy classes. A strategy class does not define behavior for each interface although it defines interfaces for executing information processing, such as algorithmInterface() in Figure 49. Exceptionally, a method in this class might be defined to invoke sub-methods provided by the same class, in cases where the method is large and complicated. This is useful as it makes it easy to define behavior of information processing. The behavior for each interface is defined in a concrete strategy class derived from the strategy class. Other behavior for the same interface can be defined in another concrete strategy class.

This group has a relationship with concrete state classes that invoke information processing in a concrete strategy class. A concrete state class delegates tasks to the concrete strategy class with which it is related. All information is transmitted from the concrete state class to the concrete strategy class. This eliminates it from the concrete state class to define additional interfaces that are used by the concrete strategy class to get information.

This object group is defined for each information processing, e.g. in Figure 49, class "Strategy A" is defined for class "Concrete State A2", and class "Strategy B" is defined for class "Concrete State B2." An instance of each concrete strategy class is created by one, and shared by several instances of state classes. A concrete state class can select appropriate behavior by changing the relationship to a concrete strategy class. This change can be done dynamically with no modification of a concrete state class. If a concrete state class does not have a pointer to a concrete strategy class, no information processing is executed.

6.3.2.3 Object group for additional data

This object group provides a way to add new data needed for information processing, but not implemented in existing classes. This object group is composed of an element class, a decorator class, concrete decorator classes, and a proxy class. The 'target class' in Figure 49 means an

existing class for general-purpose use.

An element class implements the interfaces that are defined in the target class. It does not define behavior.

A decorator class is derived from an element class and is related to the element class. This connects sequentially an instance derived from the element class with another instance derived from the element class. A concrete strategy class is not aware of the difference between a target class and a concrete decorator class by this sequential relationship.

Additional data and/or methods are defined in a concrete decorator class derived from a decorator class.

A proxy class is derived from an element class instead of a target class to which it is related. It invokes methods provided by the target class. The target class is not required to change using this mechanism.

6.3.3 Attachment of the package to an information model

One package will be attached to an information model for each requirement that needs to change a function. Therefore, the number of packages increases in proportion to the number of such requirements.

A requirement for which the package is created is obtained from dividing the whole of the requirement. This division is performed based on states that invoke, or do not invoke, information processing. The state-based division makes clear which object class an instance of the PEP should be attached to.

A target class is identified based on the contents of information processing defined in a concrete strategy class. Therefore, a full account of the information processing for a requirement that makes use of the package is needed.

6.4 Package applications

We designed models for two management functions that make use of the PEP. One makes schedules of maintenance tasks, and the other sets threshold values concerned with quality of service (QoS). Requirements of some operators in a power company are shown by these models. The operators provide electric power systems with communications services that have unique QoS. The two applications provide management functions that are different to those for general-purpose use.

6.4.1 Scheduling maintenance

Working dates are determined in making schedules of maintenance tasks, meeting the constraints of each system, or the order of the tasks. This function may differ from one operator to another, as it reflects requirements intrinsic to the operator.

The requirements related to this scheduling are categorized as follows.

1) Conditions determining if a task can be executed
2) Criteria selecting tasks to be changed
3) Criteria selecting an alternative date

The PEP was used in this application for defining behavior based on the constraints. We considered the following two constraints to illustrate how the package is used.

1) A constraint prohibiting suspension of an electric power system (EPS) composed of two communication routs
2) A constraint maintaining task order

Figure 50 depicts an information model that makes use of two packages implementing each constraint. In this section, we explain the application of the package to a constraint related to communication routes.

An EPS class is usually composed of two routes for communication, in order to ensure a high degree of reliability. As suspension of both routes interrupts the EPS, the situation requires that the task schedule be coordinated. This is modeled as the state aggregation class and state of two routes' classes. Each class derived from "State of two routes" decides the next state, and provides methods for the transition. The "Both suspension" class that shows a state requiring schedule change is related to the "Coordination for a route constraint" class.

Two concrete strategy classes derived from "Coordination for a route constraint" were designed in this example. One coordinates a schedule based on how many times a task schedule is changed. The other changes a schedule according to task priority. Both of these provide methods that have the same signature, but use different procedures.

"Change history" and "Task priority" were designed, since each coordination class requires its own additional data for the task classes. These classes provide methods for the coordination classes, and the "Task proxy" class is related to the "Task" class for general-purpose use.

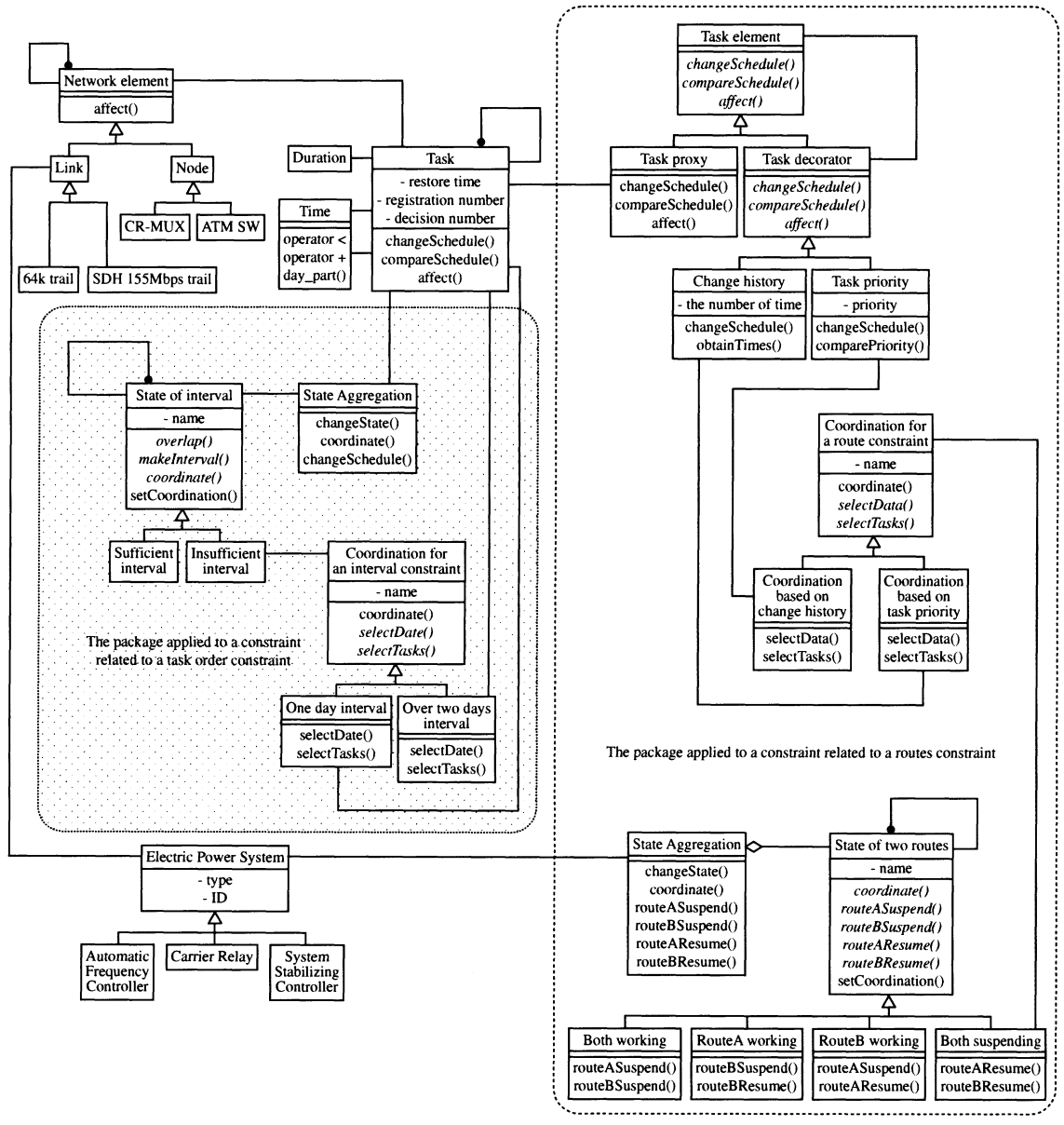


Figure 50: Information model for maintenance scheduling using the PEP

6.4.2 Setting threshold values for QoS

We designed another application in order to evaluate the performance of the package, as illustrated in Figure 51. This application sets threshold values for QoS in telecommunications networks for an electric power company. Threshold values in this application are set according to the state of an electric power system, or a power transmission system supported by it.

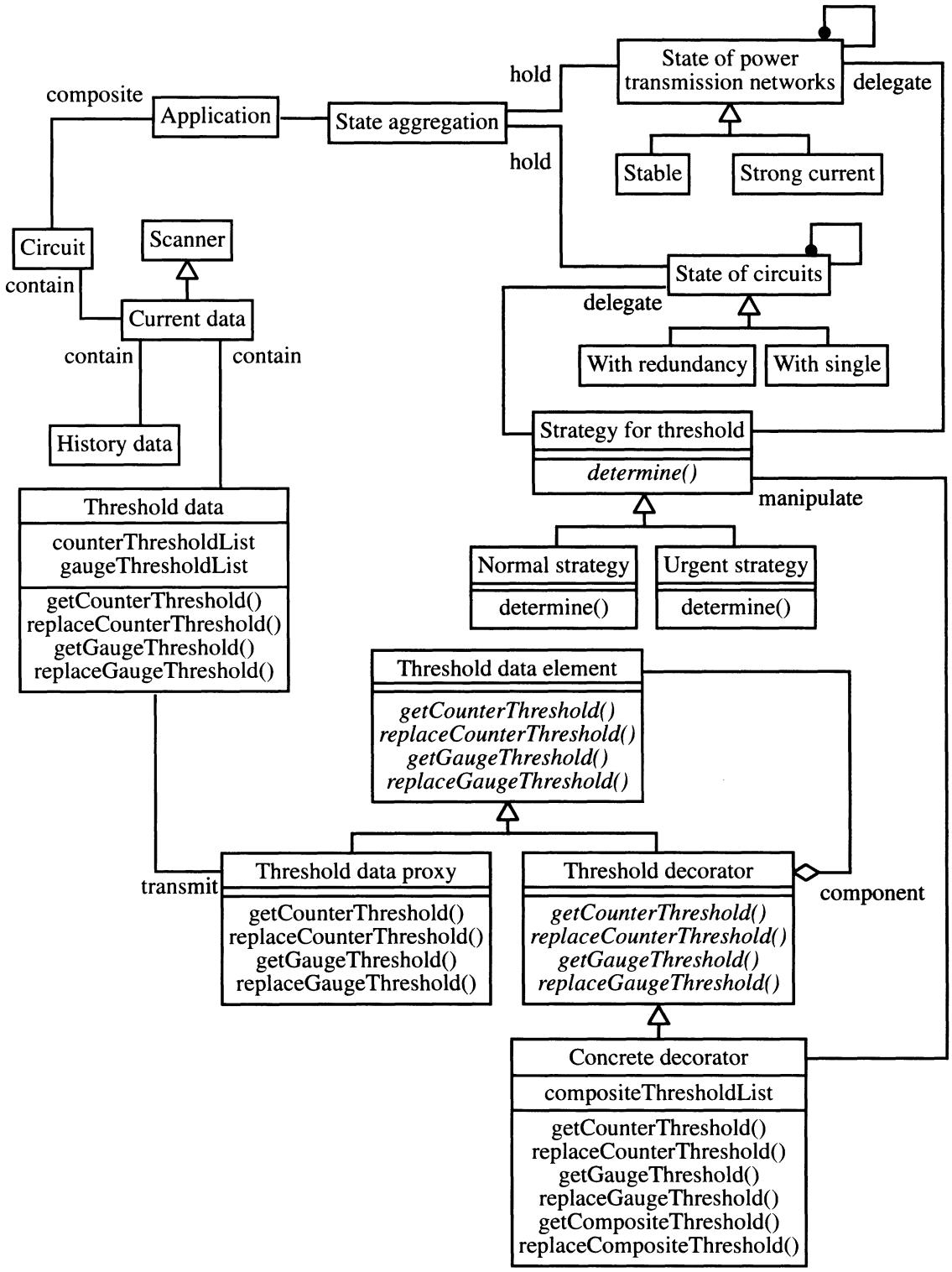


Figure 51: Information model for setting threshold values for QoS

A partial extension package in the model has a state object group that represents status to be considered with regard to thresholds. Calculations of threshold value were provided in classes derived from a strategy class. An object group for additional data in this model includes a composite threshold list that is composed of primitive counters and gauges.

6.5 Evaluation of PEP using software metrics

We made use of software metrics to evaluation the defined flexibility of the applications. In this section, we explain adopted software metrics, and show results of the evaluations.

6.5.1 Software metrics

Many metrics have been proposed for object-oriented software. Most of them measure an aspect of software quality, but do not evaluate software from all angles. Frameworks about such metrics [140, 141] have been proposed that categorize metrics and support selection of metrics in order to improve this situation. In this thesis, we adopted a framework proposed by Briand et al. [139]. This framework defines and classified metrics focused on coupling between objects.

Based on this framework, the following metrics were used to measure the performance of the packages in terms of their flexibility. RFC (Response for Class) is defined as follows [142].

> Let $R_0(c)$ be the set of methods of class c, and $R_{i+1}(c)$ be the set of methods invoked in the polymorphic manner by methods in $R_i(c)$, then
>
> $$RFC\alpha(c) = \left| \bigcup_{i=0}^{\alpha} Ri(c) \right| \text{ for } \alpha = 1,2,3, \ldots$$

This means how dependent the measured class is on other classes. In other words, it shows the reusability of the measured object class. Parameter α is set as 1 in this thesis.

OMMEC (Other Method-Method interaction for Export Coupling) [143] indicates how many times methods provided by a class to be measured are called by other classes. This shows the localization of modification in terms of the measured class. In the case of these metrics, the smaller the value is, the more flexible the PEP is.

This measurement of flexibility has several good points, as follows. We can estimate the relative performance of two information models, based on measurement values. The values can be measured at the design stage of classes. In addition, definitions of the measurement are easy to understand. Therefore, these values are suitable for assessing experiences of the package.

6.5.2 Measurement of the model for maintenance scheduling

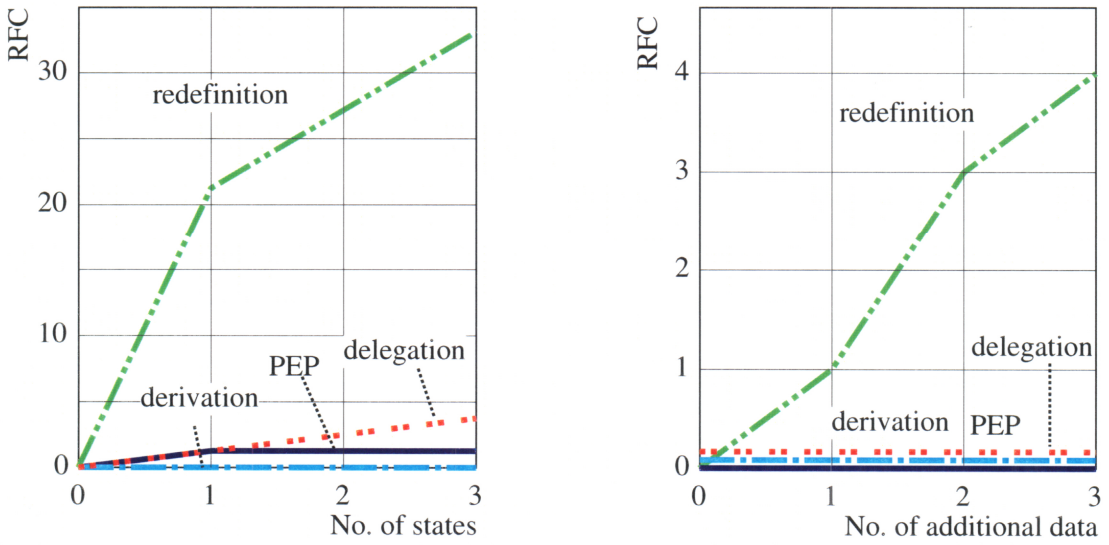
We measured the metric values of the objects for maintenance scheduling. For this measurement, new state, strategy, or additional data objects are added to existing object one by one. Some of the added objects were not shown in Figure 50.

We also measured the metric values of information models using three other customizations of functions. A class of an existing object is defined again to meet a requirement in the first customization. This is referred to as class redefinition in this thesis. A new class is derived from a class of an existing object to fit a requirement in the second customization. It is referred to as class derivation in this thesis. A new class is defined for delegated tasks from an existing object in the third customization. This is referred to as delegation in this thesis. These are major methodologies to extend the functions of an object.

For comparison between these methodologies, we show graphs concerned with RFC or OMMEC. The values in the graphs indicate differences from the values of each class before modifications.

6.5.2.1 Reusability of classes for general-purpose use

Figure 52 (a) shows the RFC values of an EPS class in Figure 50, in respect to the reusability of classes related to states when states related to the EPS class are added one by one. An EPS class



(a) Object group for states

(b) Object group for additional data

Figure 52: The number of requirements and RFC in maintenance scheduling

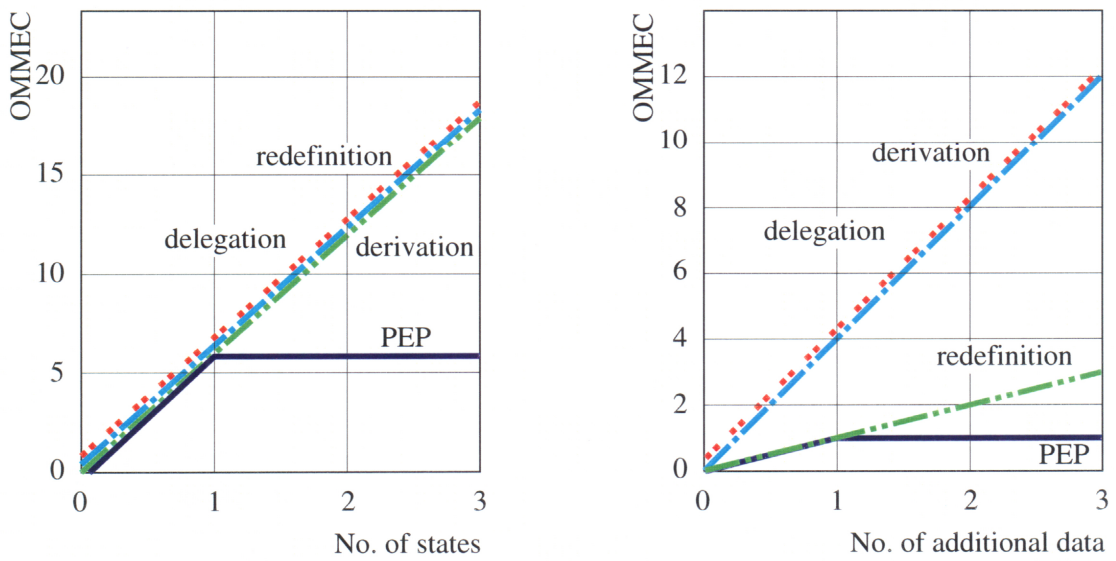
is measured in each methodology because it is designed for general-purpose use. The result shows that the first addition of a state to an EPS using the package increases its RFC value by one. The following additions do not increase the value because no method is needed to add a new state in this case. This means that the reusability of an EPS class is less affected using the package than other methodologies.

Figure 52 (b) shows RFC values of a task class in Figure 50, in respect to the reusability of existing objects relating to additional data, when data are added one by one. A measured class in each methodology is a task class as an object for general-purpose use. The result indicates that coupling between a task class and the package stays weak, even if the amount of additional data increases. The package is able to maintain the reusability of the task class.

6.5.2.2 Localization of the influence of modification

In regard to classes related to states, Figure 53 (a) shows OMMEC values of classes implementing attributes and methods related to states when states related to the EPS are added one by one. Modifications, and a measured class in each methodology, are as follows.

1) Class redefinition: We measured the EPS class to which new attributes and methods for states were added.



(a) Object group for states

(b) Object group for additional data

Figure 53: The number of requirements and OMMEC in maintenance scheduling

2) Class derivation: Attributes and methods were added to the class derived from the EPS class. This was measured because an instance of this class will be created.
3) Delegation: We measured a class to which the EPS class delegates tasks.
4) PEP: An object group for states in the PEP provides attributes and methods for states. A state aggregation class in the package was measured.

The result shows that the package increases OMMEC value at the first addition of a state. It does not however increase the value after that. This arose from that no additional methods were implemented in the state aggregation class.

In regard to classes related to additional data, Figure 53 (b) shows OMMEC values of the classes when data are added one by one. The following describes how a class was modified, and which class was measured in each methodology.

1) Class redefinition: We measured a redefined task class that implemented attributes and methods for additional data.
2) Class derivation: A class derived from a task class implemented attributes and methods for additional data. We measured the derived task class, because an instance of this class will be created.
3) Delegation: Attributes and methods for additional data were implemented in a class delegated from an EPS object. We measured the delegated class.
4) PEP: The object group for additional data provided attributes and methods. We measured a concrete decorator class.

The result shows that the concrete decorator class increases its OMMEC value by only one. In addition, this does not influence objects for general-purpose use. Therefore, it is not possible for a class to have a major influence on other classes.

Figure 54 shows OMMEC values of classes implementing information processing, against the number of requirements or the kinds of processing. Measured classes in each methodology are as follows.

1) Class redefinition: We measured an EPS object that defines new attributes and methods for information processing.
2) Class derivation: A class derived from an EPS class had new attributes and methods. We measured the derived class because an instance of this class will be created.
3) Delegation: We measured a new class to which an EPS object delegates tasks. This new class defined attributes and methods for information processing.
4) PEP: We measured the object group for information processing that defines new attributes and methods.

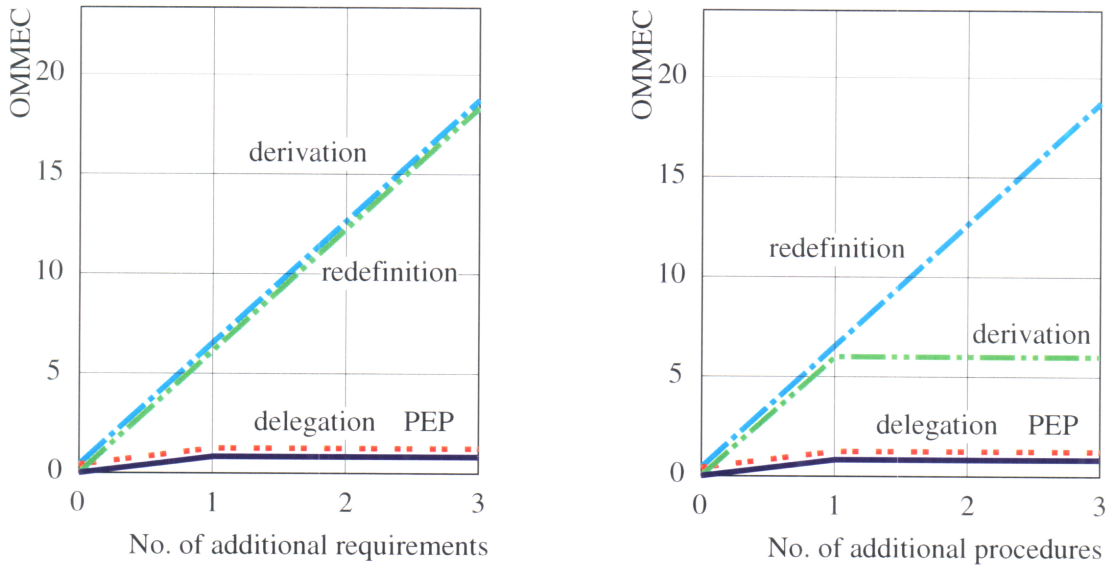


Figure 54: OMMEC of object groups for information processing in maintenance scheduling

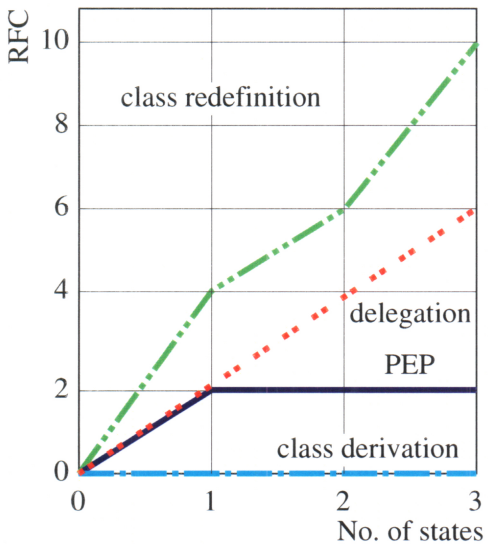
The result shows that the modifying influence of the package can be localized in the both cases.

6.5.2.3 Measurement of the model for QoS threshold

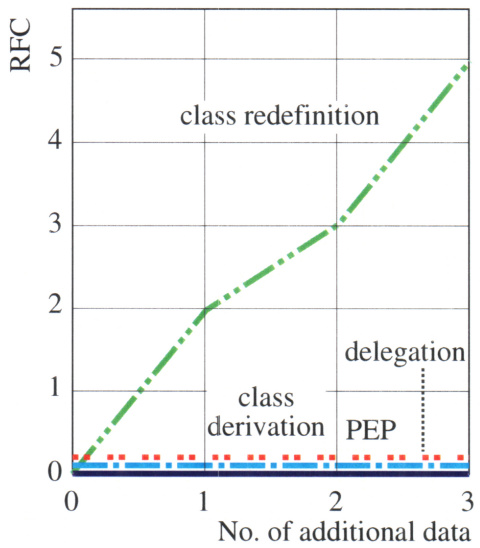
We measured object in the same manner as in maintenance scheduling.

Figure 55 depicts RFC values with regard to classes representing states or additional data in the model for QoS threshold setting. The result shows a similar trend to the model for maintenance scheduling.

Figure 56 shows OMMEC values with regard to classes representing states or additional data in the model. The main differences between this model and the model for maintenance scheduling are OMMEC values of an object group for states in the package. In this model, the addition of each state needs new events invoking methods in the state aggregation class. On the other hand, the addition of states in the model for maintenance scheduling does not need the invocation of new methods.

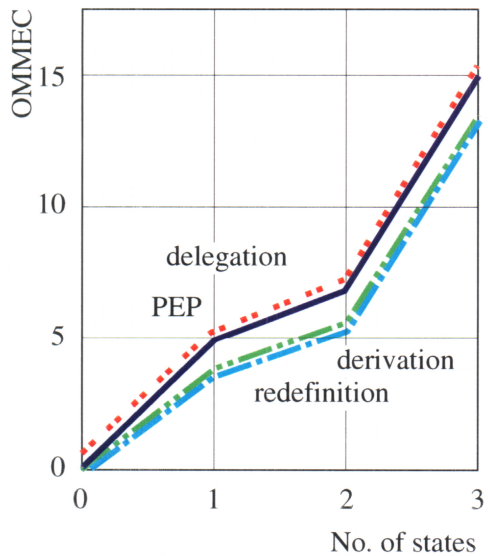


(a) Object group for states

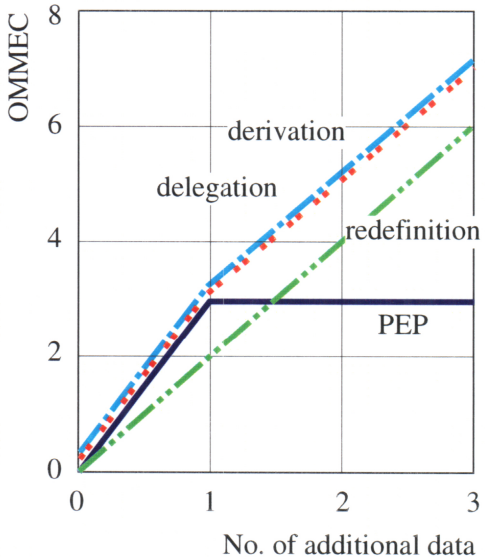


(b) Object group for additional data

Figure 55: The number of requirements and RFC in QoS threshold setting



(a) Object group for states



(b) Object group for additional data

Figure 56: The number of requirements and OMMEC in QoS threshold setting

Figure 57 depicts OMMEC values of classes that defined information processing. Most values in both of these graphs varied in a way similar to those in the maintenance scheduling application. The values of the package where requirements are added one by one are different from the model for the maintenance scheduling. This was caused by sharing an instance of the same concrete strategy class. All instances of concrete state classes, e.g. "With redundancy" class in Figure 51, invoked a method provided by a concrete strategy class, e.g. "Normal strategy" class in Figure 51. OMMEC values corresponded to the number of instances of state classes. However, these concrete strategy classes had an influence only on concrete state classes as custom objects.

6.5.2.4 Summary of the evaluation

These results indicate that the first addition of data or processing has an influence on existing objects in terms of the reusability. However, the reusability is maintained in the subsequent additions with no side effects. Although the localization of modification may deteriorate depending on the change of a function, the influence of the package is the smallest of the methodologies.

In addition, the package keeps the reusability of existing objects as well as the localization. One of them is sacrificed in other methodologies.

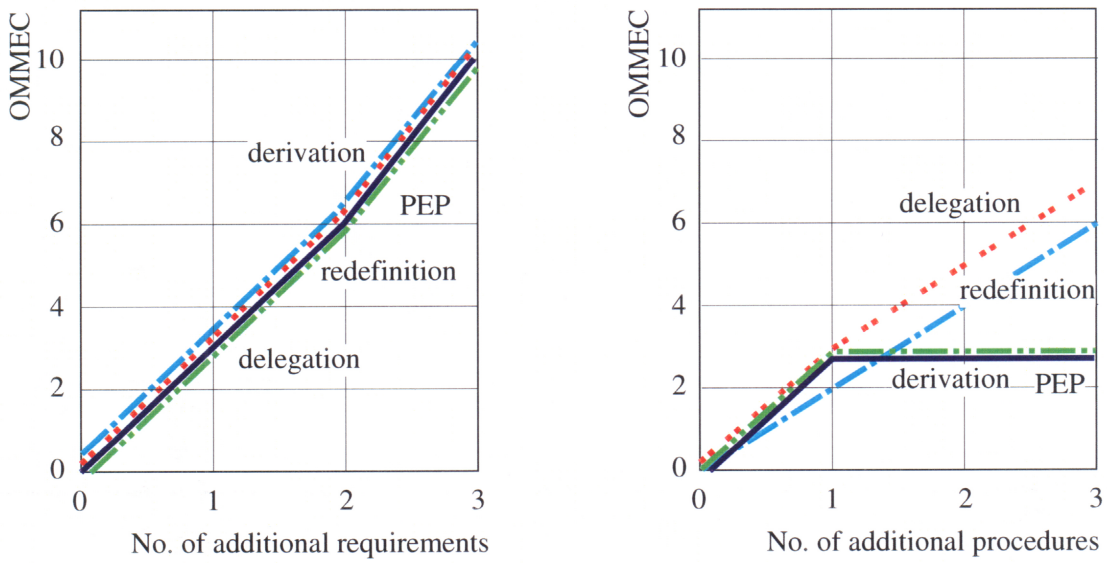


Figure 57: OMMEC of object groups for information processing in QoS threshold setting

6.6 Summary of the package

In this chapter, we have proposed a partial extension package (PEP) that adds or changes the behavior of a function. This package is composed of three object groups for states, information processing and additional data.

We developed two applications based on the package; one schedules maintenance tasks, and the other sets values of QoS. They are estimated in terms of the flexibility of using software metrics. The results show that our contribution confers benefits on the design of objects for customizing function, as mentioned below.

1) **Flexible customization making two contrary points compatible**: There are two aspects of flexibility in customizing functions. One is to keep classes for general-purpose use reusable. The other is to localize the impacts of a modification. In general, it is not easy to make them compatible, however, our contribution achieves this compatibility.

    a) **Maintenance of reusability**: PEP has two connection points to existing classes. The state aggregation class has one of the connections, and the proxy class has the other one. PEP requires an existing class to contain codes to call methods provided by the state aggregation class. It does not require that other classes include any codes. Therefore, most existing classes can keep their reusability.
    b) **Localization of the impacts of modifications**: Methods provided in the object group for information processing are called only by the object group for states. In addition, the number of method calling is minimal. Two other object groups provide minimal methods to respective classes outside of the groups. This localization contributes to the simple maintenance of classes for general-purpose use and the easy addition of a function.

2) **Simple use**: PEP can be used easily because of its granularity and structure. The following explains these two points.

    a) **Proper granularity**: PEP is larger than any design pattern, and smaller than a framework. A part on this scale has not appeared in object-oriented design. It facilitates the addition of new functions and data to existing classes. In addition, its scale is not so large to understand.
    b) **Structure with the essence of object-oriented design**: PEP contains the essence of object-oriented design, and a programmer can make use of this with only knowledge of the role of each class. It eliminates learning the object-oriented

technique. A programmer can enjoy the benefit of the object-oriented technique even though he/she is not skilful. It is important for a third-party to add a function to an NMS.

Chapter 7
Conclusions

In this thesis, we are concerned with how to support rapid and precise function design. We have focused on the domain of network management, because of the rich standards we can use and the variety of functions to be designed.

Functions provided by an NMS have the following features:

1) Some functions can be shared between different network operators, while others are specific to each operator.
2) Functions must be adaptive to conditions in networks or services.
3) Functions require expertise in order to be adaptive.
4) It is difficult to describe a sequence to process a function.
5) Requirements for a function may not appear at the time the NMS is built.

We have clarified the shortcomings of current methods in cases where functions for network management are designed. The shortcomings are as follows:

1) Network operators cannot easily tell what functions they need, using current methods for requirement engineering. On the other hand, it is also difficult for system designers to understand the operators' requests.
2) Functions may not be adaptive in an environment in which user requirements are changeable.
3) Few techniques are put forward to design individual functions based on shared software.

To address these issues, we proposed three techniques described as follows:

1) A technique for requirement elicitation and description
2) A mechanism selecting a subset of policies that meet a condition
3) Techniques for customization of off-the-shelf components and the standard object classes

In our technique for requirement elicitation and description, users draw up their requirement. We have adopted use cases detailed by policies in order to allow users to draw up their requirements. Several types of policies have been set up to represent network operations. Two formats for describing policies are also provided by our technique. This technique confers the three benefits mentioned below on requirement analysis and description.

1) **User-friendly notations**: This is brought by use case detailed by policies that are suitable for network operations. A use case is so simple that an operator can understand and specify it even though he/she is a novice in the design. In addition, a policy is useful for a network operator to give details of a use case. This is because an operator is familiar with a policy because it comes from a rule in his/her business.
2) **Facilitation of communication between users and designers**: The notations are semi-formal so that ambiguity will slip into a specification less often. Besides, a system designer understands each function declared in a policy. The volume of description for each function is simple enough to be understood easily. It also contributes facilitating communications between a user and a system designer, because the work of the designer to elicit requirements from the user is eliminated.
3) **High compatibility with object-oriented design**: This technique is based on use case diagrams. In addition, it reinforces existing approaches for system designers. Therefore, it is compatible with object-oriented design methods. This feature is important for our technique to enjoy wide acceptance.

We have developed a mechanism selecting policies in order to combine a function provided by an NMS and a policy specified as a requirement. This mechanism selects policies that fit into a condition, and invokes functions specified by selected policies. Artificial immune networks are applied to the mechanism for selecting policies. The rules for setting parameters are also provided by this technique. This technique confers the following benefits on function design or network operations.

1) **Proper outputs**: The selected policies are suitable for actual operations of networks. The mechanism permits the existence of a compromise policy and can select a suitable set in accordance with conditions in networks or services. This allows the users to skip drawing up a processing sequence in NMS. Besides, the calculations for selecting policies finish in a short time that is sufficient to be used in actual operations. Of course, the proper outputs assist the job of network operators.
2) **Adaptability to change of policies**: This policy selection allows a new policy to be added to an existing set of policies, a policy to be removed from the set or to be changed, without a large amount of labor. This is because the formulas for selection do not need to be changed when a policy is added, changed or deleted. The rules for

setting parameters are also provided in our technique. They also contribute the adaptability to a change of policies. These features come from the concept of artificial immune networks.

We have developed RevComponents and PEP (Partial Extension Package) for techniques to add new features to existing software. A RevComponent is a software component. It adds, removes and changes a function provided by an existing component. PEP can add a new feature to existing object classes with minimal influence. Existing software can change its features in a flexible and simple manner using these techniques. These two techniques confer the benefits described as follows on the customization of functions.

1) **Keeping maintainability of existing software**: RevComponents make use of interface definitions for an off-the-shelf component, and customize a function provided by the off-the-shelf component. They do not require change of source codes in the off-the-shelf component, because they intercept a message from/to the component to provide customized functions. On the other hand, PEP limits the number that an existing class calls methods provided by a class in PEP. In addition, it adds new data and procedures without a change in an existing class. As a result, PEP keeps weak coupling to the existing classes. These features contribute to the maintainability of existing software. This enables the construction of an NMS using off-the-shelf components as well as providing functions specific to an operator.
2) **Rapid customization**: Our technique provides a code generator for RevComponents. This eliminates labor in making a program for the interface of an off-the-shelf component, or message forwarding. The workload involved in creating a RevComponent is the same amount or less than modifying the off-the-shelf component. On the other hand, PEP also provides rapid customization. It has three object groups so that a designer can embed programs for states, procedure and additional data in each object group. It has a structure that enable a state, a procedure or additional data to be changed without impacts on other classes in PEP. A function can be customized rapidly using these two techniques.

Our contributions cover the range from requirement engineering to object-oriented design. The whole of our approach enables a function needed by a user to be created rapidly and cost-effectively using the standard technologies. The case studies on an NMS show that our techniques can be used in terms of output correctness, processing speed, and/or maintainability. Our contribution reduces costs and the period necessary to build an NMS.

Our contributions have three important key points; function design led by users, adaptive functions and the customization of shared software with minimal influence. We believe these points to be effective in other domains, and have suggested the domains in which our

contributions may be effective. In addition, we know of issues to be addressed for our techniques to be used more effectively. Through the development of techniques with the three key points, we would like to create a new area in which users can take part in designing functions. Existing software as valuable artifacts can be shared easily in this area.

Chapter 8

Future works

In this chapter, we discuss future work about techniques that have been proposed in this thesis. We have a number of hurdles that have to be solved in order to enhance their effects. We focus on the following six issues that should be addressed in the next stage.

1) Application of the techniques to other systems
2) Ontology for policies and function calls
3) A tool supporting FDLU
4) Validation and verification of designed policies
5) Adoption of mobile agents for dynamic change of behavior
6) Integrated CASE tool for PEP and RevComponent

We will mention the details of each issue in the following sections.

8.1 Application to other domains

We have described our techniques for rapid and precise function design for network management systems. These techniques are not limited to the domain, but may be applied to others if they fulfill several conditions. In this section, we discuss the possibility of application of the techniques to other domain.

FDLU makes use of use cases detailed by policies so that it fits to domains in which functions supporting tasks can be defined as rules. Our mechanism of policy selection also depends on the same conditions.

The RevComponents and PEP assume rich information model that are defined as standard, and off-the-shelf components compliant to the model. They can be used to modify existing objects that do not comply with some standards, but use object- and component-oriented construction.

These conditions indicate that the operation of power systems is a domain in which our techniques perform effectively. Tasks in the operations are performed in conformance to rules.

The domain has the same characteristic as network operations in which functions provided by systems are designed based on a large amount of expertise. In addition, standard information models are being developed for this domain. Some of the models are based on object-oriented technology. For example, standard models have been developed for switchgear, transformers, substation and field devices [144].

Building industry may also be a domain in which our technique may show promise. IFC (Industry Foundation Classes) has been developed as information models for CAD (Computer Aided Design) tools and other systems in the industry [145]. The models aim to assist that systems from different vendors can inter-operate without any barriers. It may be difficult to set policies for design or maintenance of buildings. Basic policies, however, might support tasks in the design or maintenance of buildings. If so, all techniques proposed in this thesis could be applied to systems in this industry. These could be powerful for the design of functions in the systems because production to order represents a large percentage in this industry.

We believe that our techniques are effective in domains besides the network operations. Especially, the viewpoint of user-oriented design will assist the rapid and precise design of functions in a system. To validate the power, we will have to concretely apply these techniques to the domains.

8.2 Ontology for policies and function calls

In FDLU, we have not yet defined ontology relating to the network operations. Users can use arbitrary terms to write conditions and functions in policies. However, this may become a barrier for a system designer to understand policies. If so, associations between a policy and a function may be incorrect.

Fortunately, we can make use of a large number of terms defined as standards for description of policies. Terms in transmission methods such as IP [146] or ATM [147] have accurate meanings. As well, network operators can make use of terminology for cables [148], and for digital transmission and multiplexing [149]. For example, "digital channel" is defined as "The means of unidirectional digital transmission of digital signals between two points" [149]. Network operators can describe policies relating to monitor or control of network elements using these terms. System designers can share the definitions of such terms relating to network elements.

Terms and definitions of traffic engineering [150] and that related to quality of service and network performance [151] are also defined as standards. For example, "network performance" is defined as "the ability of a network or network portion to provide the functions related to communications between users" [151]. These terms and definitions help a network operator to describe policies relating to his/her operations and/or services.

We will have to investigate if the standard terms are useful for associating a policy and a function. The system designers had few troubles to understand terminology in our case study. However, the number of samples is too short to validate the usefulness of the standard terms.

In addition, we may have to develop a method in which a network operator defines a new term describing his/her original service. Network operators are developing their original services day by day as well as trying to make their tasks more efficient. These may require new terms that are not defined as the standards. They currently define a new term with no assist of computer systems. This task should be improved in order to offer a new service quickly.

8.3 A tool supporting FDLU

We have not yet developed a tool that supports function design led by users (FDLU). Such a tool should provide the supporting functions that are mentioned below.

1) Derivation of policies from global purposes of a business
2) Attachment of policies to a use case
3) Conversion of representations for specifying policies

Each policy declared in the process of FDLU is derived from one or more purposes of a user's business. This thesis does not propose any new approach to the derivation. This tool may be able to make use of one or some approaches described in Section 3.2.1.2. These approaches, however, have several problems mentioned in the section so that a new technique may be needed to support the goal-oriented requirement analysis of FDLU. In the new technique, it will be the key to integrate analysis using use cases and the traditional goal-oriented approach.

Attachment of policies to a use case may be associated with the technique of supporting effective derivation of policies. In this thesis, we do not suggest what should be specified in a use case or as a policy. We will have to clarify more strictly the role of a use case and a policy in FDLU, in order to develop the supporting tool.

It is also important to convert the graphical representation for describing policies to the table representation, and vice versa. The lack of an automatic conversion of the representations inconvenienced the network operators to specify their policies, as mentioned in Section 0. This tool can enable users to draw up their requirements by themselves. Two types of such conversion tools can be considered: batch on-demand conversion and synchronized one. The on-demand conversion brings contents in a representation to another one when an event such as an explicit user command occurs. This type may enable users to manipulate the tool quickly. However, it cannot maintain consistent displays of the two representations. On the other hand, the synchronized conversion brings contents in a representation to another one each time a

policy is added, deleted, or modified. This conversion can display some contents in the different representations. However, it is estimated that the tool works slowly.

We have to pay attentions to these issues when developing such a tool.

8.4 Validation and verification of policies

To validate and verify policies that are set by network operators is an important issue to be considered. We have not set a method for validation and verification of policies in this thesis. If policies that are invalid or not verified are used for network operations, the NMS using the policies performs incorrectly and may have negative effects on the operations. This issue has been recognized widely [152].

Many techniques have been proposed for validation and verification of knowledge. Graphical techniques have been found to provide a good framework for the detection of errors that may appear in a rule base [153, 154, 155, 156, 157]. Topological properties of graphical structures are used to deduce dependencies across propositions. These graphical representations allow the verification problem to be reformulated as one of reachability of specific states in the graph.

Another important approach is testing of knowledge bases for the validation of rule-based systems. The testing complements the graph-based methods that are out of verifying and validating of knowledge specific to a certain domain. Normally, a reduced number of test cases are selected in the testing because the patterns of testing are too many to consider. Selection of test suites, therefore, is the key to validating and verifying knowledge. Much literature has mentioned how to select test suites [158, 159]. These test suites are usually applied to static or off-line tests. If the tests were performed on a real system, the system would be down or process data incorrectly.

Nevertheless, we believe that simulations on real systems are powerful for validation and verification. The field of software certification has noted the importance of dynamic testing of knowledge [160]. Dynamic testing may cover a test suite that is not experienced or forecasted so far. However, it is even more complicated because there is lack of simulators or drivers to generate realistic data input for a desired test suite. Therefore, we hope to develop a simulation scheme that runs on but does not interfere with a real NMS. We need to note user's viewpoints at the development of the simulation scheme.

8.5 Installment and removal of RevComponents without system interruption

RevComponents proposed in Chapter 5 can be plugged in or removed from an NMS that is built based on components conforming to the standards. However, they need to unbind and rebind connections to the target component when RevComponents are plugged in or removed. This interrupts execution of the NMS.

Mobile agents are one of the techniques that solve this problem. As mentioned above, a mobile agent holds data and procedures, and moves between computers [116]. This technique may allow RevComponents not to unbind and rebind the connections to the target component. We have held back adopting this technique because of its performance and difficult design. To make use of mobile agents, we have to overcome these shortcomings. In addition, we have to note the compatibility between mobile agents and off-the-shelf components.

There is only a very limited amount of literature available on the tuning performance of mobile agents. Foster has proposed a way in which the performance of mobile agents can be improved [161]. In terms of maintenance of mobile agents, we have more literature, for example, ADK (AgentBeans Development Kit) [162] and a Petri-net based framework [163].

A proxy component may also be a promising option. A proxy component intercepts all messages sent to the target component even if there is no RevComponent. This technique eliminates unbinding and rebinding a connection to the target component because the proxy is always set. However, it must delay the processing time in an NMS. Before the proxy is installed, we must test its performance.

8.6 Integrated CASE tool for PEP and RevComponents

We have proposed two techniques for implementing desired functions, RevComponents and PEP. However, we have not developed a CASE (Computer Aided Software Engineering) tool using these techniques. A CASE tool is necessary in order to make the techniques more convenient. In this section, we review what kinds of CASE tools have been developed so far, and clarify any potential issues that need to be addressed.

A POD (Pattern-Oriented Design) tool supports visual composition of design patterns [164]. Patterns can be integrated easily by means of this tool. Developers can use it to trace

patterns through various abstraction levels. The benefits of this tool have been reported as follows.

1) It supports hierarchical design.
2) It provides a mechanism to trace a pattern to an application class, and vice versa.
3) It provides designers with a pattern composition view that is a higher level of design documentation than class diagrams.

On the other hand, it has also been reported that the tool does not address the problem of how constructional patterns can be combined with parts of a design that are not expressed as patterns. The Fragmentation technique [165] and the PSiGene CASE tool [166] also help to bind one design pattern with another.

FACE (Framework Adaptive Composition Environment) sustains an incremental development style without abandoning the higher-level design pattern abstraction [167]. It guides instantiating patterns and bridges a gap between design and implementation. Patterns Wizards is another example of such a tool [168]. This tool adopts a concept of meta-programming to support the specification of design patterns and their realization in a given program.

A CASE tool for our techniques requires new features adding to that provided by tools in the literature. First, the tool should take account of standard objects. It might be effective for the tool to provide a library of the standards. Second, it should offer criteria for creative use of the two techniques. We will have to consider integration of the techniques for these criteria to be achieved. Third, the tool should hold user's viewpoint described in Section 2.3, and help collaborations between users and developers. If so, it may help to design functions more quickly and precisely.

Bibliography

[1] I. Jacobson, *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addison–Wesley, Boston, 1994.

[2] Object Management Group, *OMG Unified Modeling Language Specification, Version 1.4.* September 2001.

[3] T. Otani and Y. Yamamoto, "A Policy Selection for Network Management based on An Artificial Immune Network," to appear in *IPSJ Journal*, (in Japanese).

[4] T. Otani and Y. Yamamoto, "A Customization Method for Network Management Functions without Modification of Off-the-shelf Components," *Proceedings of IEEE ICSM 2001*, pp. 460–469, Florence, Italy, November 2001.

[5] T. Otani and Y. Yamamoto, "Partial Extension Package for the Flexible Customization of a Network Management Information Model," *IEICE Transactions on Communications*, Vol. E84–B, No. 7, pp. 1897–1906, July 2001.

[6] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process.* Addison–Wesley, New York, January 1999.

[7] TeleManagement Forum, *Telecom Operations Map, GB910 Approved Version 2.1.* March 2000.

[8] ITU-T, *Management Framework for Open Systems Interconnection (OSI) for CCITT Applications, Recommendation X.700.* September 1992.

[9] ITU-T, *Principles for a Telecommunications Management Network, Recommendation M.3010.* February 2000.

[10] TeleManagement Forum, *Service Quality Management Business Agreement, TMF506 Public Evaluation Version 1.5.* February 2001.

[11] TeleManagement Forum, *Peer-to-Peer Service Configuration Business Agreement, NMF502 Issue 1.0.* April 1997.

[12] TeleManagement Forum, *SMART Ordering SP to SP Interface Business Agreement, NMF504 Issue 1.00.* September 1997.

[13] NTT (eds.), *Control and Access Plant Networks (CAPNET).* NTT Technical Requirements, The Telecommunications Association, Tokyo, Japan (in Japanese).

[14] Telcordia, *Operations application message: language for operations application messages, Telcordia GR-831-CORE, Issue 1.* November 1996.

[15] W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2.* Addison–Wesley, Boston, MA, USA, January 1999.

[16] Object Management Group, *The Common Object Request Broker: Architecture and Specification, Revision 2.6.* December 2001.

[17] L. Pete (eds.) *Big Book of MPLS (Multiprotocol Label Switching) RFCs.* Morgan Kaufmann Publishers, December 2000.

[18] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," *IETF RFC2475*, December 1998.

[19] TeleManagement Forum, *TeleManagement Forum Strategic Plan Period 2002 – 2004, GB912, Version 4.6.* July 2001.

[20] P. Loucopoulos and V. Karakostas, *System Requirements Engineering.* McGraw–Hill, New York, June 1995.

[21] H. F. Hofmann and F. Lehner, "Requirement Engineering as a Success Factor in Software Projects," *IEEE Software*, Vol. 18, No. 4, pp. 58–66, July/August 2001.

[22] C. Jones, *Applied Software Measurement: Assuring Productivity and Quality.* McGraw–Hill, New York, June 1996.

[23] C. C. Hayes, "Agents in a Nutshell – A Very Brief Introduction," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 11, No. 1, pp. 127–132, January/February 1999.

[24] V. R. Lesser, "Cooperative Multiagent Systems: A Personal View of the State of the Art," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 11, No. 1, pp. 133–142, January/February 1999.

[25] O. Akashi, T. Sugawara, K. Murakami, M. Maruyama, and N. Takahashi, "Multiagent-based Cooperative Inter-AS Diagnosis in ENCORE," *Proceedings of IEEE/IFIP NOMS 2000*, pp. 521–534, Hawaii, April 2000.

[26] C. S. Hood and C. Ji, "Intelligent Agents for Proactive Fault Detection," *IEEE Internet Computing*, Vol. 2, No. 2, pp. 65–72, May/June 1998.

[27] M. Cheikhrouhou, P. Conti, and J. Labetoulle, "Automatic Configuration of PVCs in ATM Networks with Software Agents," *Proceedings of IEEE/IFIP NOMS 2000*, pp. 535–548, Hawaii, April 2000.

[28] C. Melchiors and L. M. R. Tarouco, "Troubleshooting Network Faults Using Past Experience," *Proceedings of IEEE/IFIP NOMS 2000*, pp. 549–548, Hawaii, April 2000.

[29] F. Hayes-Roth, "Rule Based Systems," *Communications of the ACM*, Vol. 28, No. 9, pp. 921–932, September 1985.

[30] J. P. Ignizio, *Introduction to Expert Systems: The Development and Implementation of Rule-Based Systems.* McGraw–Hill, New York, January 1991.

[31] M. Sloman, "Policy Driven Management for Distributed Systems," *Journal of Network and Systems Management*, Vol. 2, No. 4, pp. 333–360, Kluwer Academic, New York, December 1994.

[32] TeleManagement Forum, *Performance Reporting Concepts & Definitions, TMF701, Public Version 2.0.* November 2001.

[33] TeleManagement Forum, *SLA Management Handbook, GB917, Public Evaluation / Version 1.5*. June 2001.

[34] ITU-T, *Structure of Management Information: Management Information Model, Recommendation X.720*. January 1992.

[35] ITU-T, *Structure of Management Information: Definition of Management Information, Recommendation X.721*. February 1992.

[36] ITU-T, *Generic Network Information Model, Recommendation M.3100*. July 1995.

[37] ITU-T, *GDMO engineering viewpoint for the generic network level model, Recommendation G.855.1*. March 1999.

[38] ITU-T, *Synchronous Digital Hierarchy (SDH) – Management Information Model for the Network Element View, Recommendation G.774*. February 2001.

[39] ITU-T, *Asynchronous Transfer Mode Management of the Network Element View, Recommendation I.751*. March 1996.

[40] TeleManagement Forum, "System Integration Map: Version 2," *Current UML Models by Name*, http://www.tmfcentral.com/kc/repository/warehouse/Components/sim2/SimCatalog.html.

[41] TeleManagement Forum, "World Ordering: Version 3," *Current UML Models by Name*, http://www.tmfcentral.com/kc/repository/warehouse/Components/wot3/index.html.

[42] K. Kim, D. Lee, E. Ha, J. Park, J. Kim, and S. Kim, "Provision of Global Number Portability Using CORBA," *Proceedings of IEEE/IFIP NOMS 2000*, pp. 17–30, Hawaii, April 2000.

[43] K. Nishiki, K. Yoshida, M. Oota, and M. Ooba, "Integrated Management Architecture based on CORBA," *Proceedings of IEEE/IFIP NOMS 2000*, pp. 3–16, Hawaii, April 2000.

[44] D. R. Seligman, "Managing Modems by Periodic Polling," *Proceedings of IEEE/IFIP IM '99*, pp. 531–544, Boston, MA, May 1999.

[45] W. Ng, A. D. Jun, H. K. Chow, R. Boutaba, and A. Leon-Garcia, "MIBlets: A Pravtical Approach to Virtual Network Management," *Proceedings of IEEE/IFIP IM '99*, pp. 201–215, Boston, MA, May 1999.

[46] E. Pitt, K. McNiff, and K. McNiff, *Java.rmi: The Remote Method Invocation Guide*. Addison–Wesley, Boston, MA, July 2001.

[47] Object Management Group, *Java™ language to IDL Mapping*, ptc/00-01-06, June 1999.

[48] B. Krishnamurthy and J. Rexford, *Web Protocols and Practice: http/1.1, Networking Protocols, Caching, and Traffic Measurement*. Addison–Wesley, Boston, MA, May 2001.

[49] H. Ku, J. Forslow, and J. Park, "Web-Based Configuration Management Architecture for Backbone Router Networks," *Proceedings of IEEE/IFIP NOMS 2000*, Hawaii, pp. 173–186, April 2000.

[50] R. Rong, D. Brooks, G. Fu, and E. Eichen, "Web-Based Expert System for Automated

DSL Loop Qualification," *Proceedings of IEEE/IFIP NOMS 2000*, Hawaii, pp. 201–214, April 2000.

[51] http://www.tmfcentral.com/browse.asp?CatID=580

[52] http://www.tmforum.org/browse.asp?catID=337

[53] C. Booth eds. *The New IEEE Standard Dictionary of Electric and Electronics Terms: Fifth Edition.* IEEE, 1993.

[54] Y. Shin and A. Ohnishi, "A Visual Programming Method for Developing Sequence Controller Programs," *Proceedings of Sixth Asia Pacific Software Engineering Conference*, Takamatsu, Japan, pp. 118–125, December 1999.

[55] L. V. Zul, D. Mitton, and S. Crosby, "A Tool for Graphical Network Modeling and Analysis," *IEEE Software*, Vol. 9, No. 1, pp. 47–54, January 1992.

[56] V. Shen, "Quality from Both Developer and User Viewpoints," *IEEE Software*, Vol. 6, No. 5, page 84 and 100, September 1989.

[57] T. Yamaoka, K. Tsujino, T. Yoshida, and S. Nishida, "Supporting Mutual Understanding in Collaborative Design Project," *Proceedings of Third Asian Pacific Computer & Human Interaction*, Kanazawa, Japan, pp. 132–138, July 1998.

[58] L. G. Williams, "Assessment of Safety-Critical Specifications," *IEEE Software*, Vol. 11, No. 1, pp. 51–60, January 1994.

[59] K. Go and N. Shiratori, "A decomposition of a Formal Specification: An Improved Constraint-Oriented Method," *IEEE Transactions on Software Engineering*, Vol. 25, No. 2, pp. 258–273, February 1999.

[60] S. A. Ehikioya and K. Barker, "A Formal Specification Strategy for Electronic Commerce," *Proceedings of IDEAS 97, Montreal*, Canada, pp. 201–210, August 1997.

[61] H. Giese, J. Graf, and G. Wirtz, "Modeling Distributed Software Systems with Object Coordination Nets," *Proceedings of PDSE '98*, Kyoto, Japan, pp. 39–50, April 1998.

[62] B. W. Boehm, "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, Vol. 21, No. 5, pp. 61–72, May 1988.

[63] W. W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques." *Proceedings of IEEE ICSE '87*, Monterey, CA, USA, pp.328–338, March/April 1987, originally published in Proceedings of WESCON, 1970.

[64] I. Graham and P. L. Jones, *Expert Systems: Knowledge, Uncertainty and Decision.* Chapman & Hall, London, UK, April 1988.

[65] Y. Lincoln and E. Guba, *Naturalistic Inquiry.* Sage, Thousand Oaks, CA, USA, May 1985.

[66] J. Bubenko, C. Rolland, P. Loucopoulos, and V. De Antonellis, "Facilitating 'Fuzzy to Formal' Requirements Modeling," *Proceedings of IEEE ICRE '94*, Colorado Springs, CO, USA, pp. 154–158, April 1994.

[67] A. Dardenne, A. V. Lamsweerde, and S. Fickas, "Goal-directed Requirements

Acquisition," *Science of Computer Programming*, Vol. 20, No. 1-2, pp. 3–50, April 1993.

[68] A. I. Anton, "Goal-Based Requirements Analysis," *Proceedings of ICRE '96*, Colorado Springs, CO, USA, pp. 136–144, April 1996.

[69] R. Darimont, A. V. Lamsweerde, "Formal Refinement Patterns for Goal-Driven Requirements Elaboration," *Proceedings of Fourth ACM SIGSOFT Symposium on the Foundations of Software Engineering*, San Francisco, CA, USA, pp. 179–190, October 1996.

[70] A. V. Lamsweerde and E. Letier," handling Obstacles in Goal-Oriented Requirements Engineering," *IEEE Transactions on Software Engineering*, Vol. 26, No. 10, pp. 978–1005, October 2000.

[71] C. Rolland, C. Souveyrt, and C. B. Achour, "Guiding Goal Modeling Using Scenarios," *IEEE Transactions on Software Engineering*, Vol. 24, No. 12, pp. 1055–1071, December 1998.

[72] C. Rolland, G. Grosz, and R. Kla, "Experience with Goal-Scenario Coupling in Requirement Engineering," *Proceedings of IEEE RE '99*, Limerick, Ireland, pp. 74–83, June 1999.

[73] C. Potts, "Fitness for Use: The System Quality That Matters Most," *Proceedings of the Third International Workshop on Requirements Engineering: Foundation of Software Quality REFSQ '97*, Barcelona, Spain, pp. 15–28, June 1997.

[74] C. Potts, K. Takahashi, and A. I. Anton, "Inquiry-based Requirement Analysis," *IEEE Software*, Vol. 11, No. 2, pp. 21–32, March 1994.

[75] P. Hsia, J. Samuel, J. Gao, D. Kung, Y. Toyoshima, and C. Chen, "Formal Approach to Scenario Analysis," *IEEE Software*, Vol. 11, No. 2, pp. 33–41, March 1994.

[76] A. G. Sutcliffe, N. A. M. Maiden, S. Minocha, and D. Manuel, "Supporting Scenario-Based Requirements Engineering," *IEEE Transactions on Software Engineering*, Vol. 24, No. 12, pp. 1074–1088, December 1998.

[77] B. Dano, H. Briand, and F. Barbier, "An Approach Based on the Concept of Use Case to Produce Dynamic Object-Oriented Specifications," *Proceedings of IEEE RE '97, Annapolis*, MD, USA, pp.54–64, January 1997.

[78] E. Kavakli, P. Loucopoulos, and D. Filippidou, "Using Scenarios to Systematically Support Goal-Directed Elaboration for Information System Requirements," *Proceedings of IEEE ECBS '96*, Friedrichshafen Germany, pp. 308–314, March 1996.

[79] T. Menzies, B. Nuseibeh, and S. Waugh, "An Empirical Investigation of Multiple Viewpoint Reasoning in Requirements Engineering," *Proceedings of IEEE RE '99*, Limerick Ireland, pp. 100–110, June 1999.

[80] D. T. Ross, "Applications and Extensions to SADT," *IEEE Computer*, Vol. 18, No. 4, pp. 18–34, April 1985.

[81] R. K. Stamper, "Social Norms in Requirements Analysis: an Outline of MEASUR," M.

Jirotka and J. Goguen eds. *Requirements Engineering: technical and Social Issues*, Academic Press, San Diego, CA, USA, pp. 107–139, 1994.

[82] B. Nuseibeh, J. Kramer, and A. C. W. Finkelstein, "A Framework for Expressing the Relationships between Multiple Views in Requirements Specification," *IEEE Transactions on Software Engineering*, Vol. 20, No. 10, pp. 760–773, October 1994.

[83] M. Goedicke, T. Meyer, and G. Taentzer, "ViewPoint-oriented Software Development by Distributed Graph Transformation: Towards a Basis for Living with Inconsistencies," *Proceedings of IEEE RE '99*, Limerick, Ireland, pp. 92–99, June 1999.

[84] P. P. Chen, "The Entity-Relationship Model – Toward a Unified View of Data," *ACM Transactions on Database Systems*, Vol. 1, No. 1, pp. 9–36, March 1976.

[85] ITU-T, *Formal description techniques (FDT) – Message Sequence Chart Recommendation Z.210*. November 1999.

[86] B. Regnell, K. Kimbler, and A. Wesslen, "Improving the Use Case Driven Approach to Requirements Engineering," *Proceedings of IEEE RE '95*, York, UK, pp. 40–47, March 1995.

[87] B. Regnell, M. Anderson, and J. Bergstrand, "A Hierarchical Use Case Model with Graphical Representation," *Proceedings of IEEE ECBS '96*, Friedrichshafen, Germany, pp. 270–277, March 1996.

[88] B. Regnell, P. Runeson, and C. Wohlin, "Towards Integration of Use Case Modelling and Usage-based Testing," *The Journal of Systems and Software*, Vol. 50, No. 2, Elsevier Science, New York, NY, USA, pp. 117–130, February 2000.

[89] W. J. Lee, S. D. Cha, and Y. R. Kwon, "Integration and Analysis of Use Cases Using Modular Petri Nets in Requirements Engineering," *IEEE Transactions on Software Engineering*, Vol. 24, No. 12, pp. 1115–1130, December 1998.

[90] W. Reisig and G. Rozenberg, "lectures on Petri Nets I: basic Models," *Lecture Notes in Computer Science 1491*, Springer–Verlag, Heidelberg, Germany, 1998.

[91] K. Jensen, "Coloured Petri Nets – Basic Concepts, Analysis Methods and Practical Use, Vol. 1: Basic Concepts," *EATCS Monographs on Theoretical Computer Science*, Springer–Verlag, Heidelberg, 1992.

[92] A. Virmani, J. Lobo, and M Kohli, "Netmon: Network Management for the SARAS softswitch," *Proceedings of IEEE/IFIP NOMS 2000*, Hawaii, pp. 803–816, April 2000.

[93] T. Koch, C. Krell, and B. Kramer, "Policy Definition Language for Automated Management of Distributed Systems," *Proceedings of IEEE SMW '96*, Toronto, Canada, pp. 55–64, June 1996.

[94] D. Trcek, "Security Policy Management for Networked Information Systems," *Proceedings of IEEE/IFIP NOMS 2000*, Hawaii, pp. 817–831, April 2000.

[95] E. Lupu and M. Sloman, "Conflicts in Policy-Based Distributed Systems Management," *IEEE Transactions on Software Engineering*, Vol. 25, No. 6, pp. 852–869, June 1999.

[96] I. Jacobson, M. Ericsson, and A. Jacobson, *The Object Advantage: Business Process Reengineering with Object Technology.* Addison–Wesley, Boston, MA, USA, June 1995.

[97] S. Lilly, "Use Case Pitfalls: Top 10 Problems from Real Projects Using Use Cases," *Proceedings of TOOLS USA '99*, Santa Barbara, CA, USA, pp. 174–183, August 1999

[98] D. G. Firesmith, "Use Case Modeling Guidelines," *Proceedings of TOOLS USA '99*, Santa Barbara, CA, USA, pp. 184–193, August 1999.

[99] S. Nishihara, "Fundamentals and Perspectives of Constraint Satisfaction Problems," *Journal of JSAI*, Vol. 12, No.3, pp. 3–10, May 1997 (in Japanese).

[100] S. Haykin, *Neural Networks: A Comprehensive Foundation.* Prentice Hall, Upper Saddle River, NJ. USA, July 1998.

[101] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications.* Morgan Kaufmann, January 1998.

[102] Y. Tohma and M. Abe, "Improvement of MTTF of Feedforward Neural Networks by Applying Re-Learning," *IEICE Transactions on Information and Systems*, Vol. J82–D–I, No. 12, pp. 1379–1386, December 1999 (in Japanese).

[103] S. Shimada and Y. Anzai, "Improving Adaptability of Reinforcement Learning Systems to Dynamic Environment by Decomposing and Reusing Macro-Operations," *IEICE Transactions on Information and Systems*, Vol. J84–D–I, No. 7, pp. 1076–1088, July 2001 (in Japanese).

[104] T. Obata and M. Hagiwara, "A Color Poster Creating Support System to Reflect Kansei," *IPSJ Journal*, Vol. 41, No. 3, March 2000 (in Japanese).

[105] M. Munetomo, Y. Takai, and Y. Sato, "A Dynamic Load Balancing Scheme Using a Genetic Algorithm with Stochastic Learning," *IPSJ Journal*, Vol. 36, No. 4, April 1995.

[106] T. Tada, *Men-eki no imiron.* Seidosha, Tokyo, Japan, April 1993 (in Japanese).

[107] K. Nishiyama, *Era of Immune Networks.* Japan Broadcasting Publishing, Tokyo, Japan, May 1995 (in Japanese).

[108] N. K. Jerne, "Idiotypic Networks and Other Preconceived Ideas," *Immunological Review*, Vol. 79, pp. 5–24, 1984.

[109] A. Ishiguro, Y. Watanabe, T. Kondo, and Y. Uchikawa, "Construction of a Decentralized Consensus-Making Network Based on the Immune System Application to an Action Arbitration for an Autonomous Mobile Robot," *SICE Journal*, Vol. 33, No. 6, pp. 524–532, June 1997 (in Japanese).

[110] A. Ishiguro, T. Kondo, Y. Watanabe, Y. Shirai, and Y. Uchikawa, "An Evolutionary Construction of Immune Network-Based Behavior Arbitration Mechanism for Autonomous Mobile Robot," *Transactions on IEEJ*, Vol. 117–C, No. 7, pp. 865–873, July 1997 (in Japanese).

[111] J. Suzuki and Y. Yamamoto, "Biologically-Inspired Autonomous Adaptability in a

Communication Endsystem: An Approach Using an Artificial Immune Network," *IEICE Transactions on Information & Systems*, Vol. E84–D, No. 12, pp. 1782–1789, December 2001.

[112] K. Matsumura, "Negotiation Support Agent Based on Immune Algorithm," *Transactions on IEEJ*, Vol. 119–C, No. 10, October 1999 (in Japanese).

[113] G. Glodszmilt and Y. Yemini, "Delegated Agents for Network Management," *IEEE Communications Magazine*, pp. 66–70, March 1998.

[114] A. Vassila, G. Pavlou, and G. Knight, "Active Objects in TMN," *Proceedings of IEEE/IFIP IM '97*, San Diego, CA, USA, pp. 139–150, May 1997.

[115] M. Suzuki, Y. Kiriha, and S. Nakai, "Delegation Agents: Design and Implementation," *Proceedings of IEEE/IFIP IM '97*, San Diego, CA, USA, pp. 742–751, May 1997.

[116] V. A. Pham and A. Karmouch, "Mobile Software Agents: An Overview," *IEEE Communications Magazine*, Vol. 36, No. 6, pp. 26–37, July 1998.

[117] M. Feridun, W. Kasteleijn, and J. Krause, "Distributed Management with Mobile Components," *Proceedings of IEEE/IFIP IM '99*, pp. 857–870, Boston, MA, USA, May 1999.

[118] A. Liotta, G. Knight, and G Pavlou, "Modelling Network and System Monitoring over the Internet with Mobile Agents," *Proceedings of IEEE/IFIP NOMS '98*, New Orleans, LA, USA, pp. 302–312, February 1998.

[119] Sun Microsystems Inc., *Java™ Management Extensions Instrumentation and Agent Specification, v1.0.* July 2000.

[120] M. Mezini and K. Lieberherr, "Adaptive Plug-and-Play Components for Evolutionary Software Development," *Proceedings of OOPSLA '98*, Vancouver, Canada, pp. 71–116, October 1998.

[121] M. Suzuki, H. Maeomichi, N. Shiraishi, and Y. Kiriha, "Active Q Adaptor for Programmable End-to-End Network Management Systems," *IEICE Transactions on Communications*, Vol. E82–B, No. 11, pp. 1761–1769, November 1999.

[122] M. Wegdam and A. van Halteren, "Experiences with CORBA interceptors," *Proceedings of IFIP/ACM RM 2000*, New York, USA, April 2000.

[123] T. Fraser, L. Badger, and M. Feldman, "Hardening COTS Software with Generic Software Wrappers," *Proceedings of IEEE S&P '99*, Oakland, CA, USA, pp.2–16, May 1999.

[124] P. Sewell and J. Vitek, "Secure Composition of Untrusted Code: Wrappers and Causality Types," *Proceedings of IEEE CSFW'00*, Cambridge, England, pp. 269–284, July 2000.

[125] J. Epstein, L. Thomas, and E. Monteith, "Using Operating System Wrappers to Increase the Resiliency of Commercial Firewalls," *Proceedings of IEEE ACSAC '00*, New Orleans, Louisiana, pp.236–245, December 2000.

[126] A. V. Hense, "Wrappers Semantics of an Object-Oriented Programming language with

State," *Proceedings of Theoretical Aspects of Computer Software*, pp. 548–568, September 1991.

[127] A. H. F. Laender, A. S. da Silva, P. B. Golgher, B. Ribeiro-Neto, I. M. R. Evangelista-Filha, and K. V. Magalhaes, "The Debye Environment for Web Data Management," *IEEE Internet Computing*, pp. 60–69, July/August 2002.

[128] L. Liu, C. Pu, and W. Han, "XWRAP: An XML-Enabled Wrapper Construction System for Web Information Source," *Proceedings of 16th IEEE ICDE '00*, San Diego, CA, pp. 611–621, March 2000.

[129] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison–Wesley, New York, NY, USA, January 1995.

[130] A. Hutt (eds.), *Object Analysis and Design – Comparison of Methods*. John Wiley & Sons Inc. New York, NY, USA, June 1994.

[131] R. Martin, *Designing Object Oriented C++ Applications Using The Booch Method.* Prentice Hall, Englewood Cliffs, NJ, USA, February 1995.

[132] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenson, *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, NJ, USA, January 1991.

[133] L. Prechelt, B. Unger, W. F. Tichy, P. Brossler, and L. G. Votta, "A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions," *IEEE Transactions on Software Engineering*, Vol. 27, No. 12, pp. 1134–1144, December 2001.

[134] D. Buchs and N. Guelfi, "A Formal Specification Framework for Object-Oriented Distributed Systems," *IEEE Transactions on Software Engineering*, Vol. 26, No. 7, pp. 635–652, July 2000.

[135] A. Birrer and T. Eggenschwiler, "Frameworks in the Financial Engineering Domain: An Experiment Report," *Proceedings of ECOOP '93*, Kaiserslautern, Germany, pp. 21–35, July 1993.

[136] W. Stevens, G. Myers, and L. Constantine, "Structured design," *IBM System Journal*, pp. 115–139, 1974.

[137] P. Coad and E. Yourdon, *Object-Oriented Design*. Prentice Hall, Upper Saddle River, NJ. USA, January 1991.

[138] S. Chidamber and C. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, pp. 476–493, June 1994.

[139] L. Briand, J. Daly, and J. Wust, "A Unified Framework for Coupling Measurement in Object-Oriented Systems," *IEEE Transactions on Software Engineering*, Vol. 25, No. 1, pp. 91–121, January/February 1999.

[140] J. Eder, G. Kappel, and M. Schrefl, "Coupling and Cohesion in Object-Oriented Systems," *Technical Report, University of Klagenfurt,* 1994. Also available ftp://ftp.ifs.uni-linz.ac.at/pub/ publications/1993/0293.ps.gz

[141] M. Hitz and B. Montazeri, "Measuring Coupling and Cohesion in Object-Oriented

Systems," *Proceedings of ISACC '95*, Monterrey, Mexico, pp. 10–21, October 1995.

[142] N. Churcher and M. Shepperd, "Towards a Conceptual Framework for Object Oriented Software Metrics," *ACM SIGSOFT Software Engineering Notes*, Vol. 20, No. 2, pp. 69–76, March 1995.

[143] L. Briand, P. Devanbu, and W. Melo, "An Investigation into Coupling Measures for C++," *Proceedings of ICSE '97*, Boston, MA, USA, pp. 412–421, May 1997.

[144] T. L. Saxton, "Reference Architecture for TC 57: Draft Revision 4," IEC TC57, November 2000.

[145] International Alliance for Interoperability, *IFC 2x: Model Implementation Guide Version 1.0*. June 2001.

[146] G. Malkin (eds.), "Internet Users' Glossary," *IETF RFC 1983*, August 1996.

[147] ITU-T, *Vocabulary of Terms for Broadband Aspects of ISDN, Recommendation I.113*. June 1997.

[148] ITU-T, *Terminology for cables, Recommendation G.601*. November 1988.

[149] ITU-T, *Vocabulary of Digital Transmission and Multiplexing, and Pulse Code Modulation (PCM) Terms, Recommendation G.701*. March 1993.

[150] ITU-T, *Terms and Definitions of Traffic Engineering, Recommendation E.600*. March 1993.

[151] ITU-T, *Terms and Definitions Related to Quality of Service and Network Performance Including Dependability, Recommendation E.800*. August 1994.

[152] W. Tsai, R. Vishnuvajjala, and D. Zhang, "Verification and Validation of Knowledge-Based Systems," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 11, No. 1, pp. 202–212, January/February 1999

[153] D. L. Nazareth, "Investigating the Applicability of Petri Nets for Rule-Based System Verification," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 4, No. 3, pp. 402–415, June 1993

[154] G. Valiente, "Verification of Knowledge Base Redundancy and Subsumption using Graph Transformations," *International Journal on Expert Systems*, Vol. 6, No. 3, pp. 341–355, 1993.

[155] R. Agarwal and M. Tanniru, "A Petri Net Based Approach for Verifying the Integrity of Production Systems," *International Journal of Man-Machine Studies*, Vol. 36, No. 3, pp. 447–468, May 1993.

[156] T. A. Nguyen, "Verifying Consistency of Production Systems," *Proceedings of the Third IEEE CAIA*, Kissimmee, FL, pp. 4–8, February 1987.

[157] M. Ramaswamy, S. Sarkar, and Y. Chen, "Using Directed Hypergraphs to Verify Rule-Based Expert Systems," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 9, No. 2, pp. 221–237, March/April, 1997.

[158] G. W. Rosenwald and C. Liu, "Rule-Based System validation through Automatic

Identification of Equivalence Classes," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 9, No. 1, pp. 24–31, January/February 1997.

[159] T. Menzies and B. Cukic, "On the Sufficiency of Limited Testing for Knowledge Based Systems," *Proceedings of IEEE ICTAI '99*, Chicago, IL, USA, pp. 431–440, November 1999.

[160] A. I. Vermesan, "Software Certification for Industry – Verification and Validation Issues in Expert Systems," *Proceedings of IEEE DEXA '98*, Vienna, Austria, pp. 3–14, August 1998.

[161] S. S. Foster, B. A. Nebesh, D. Moore, and M. J. Flester, "Performance Tuning Mobile Agent Workflow Applications," *Proceedings of TOOLS '99*, Santa Barbara, CA, USA, pp. 8–17, August 1999.

[162] T. Gschwind, M. Feridun, and S. Pleisch, "ADK – Building Mobile Agents for Network and Systems Management from Reusable Components," *Proceedings of ASA/MA '99*, Palm Springs, CA, USA, pp. 13–21, October 1999.

[163] O. F. Rana, "A Design and Management Framework for Mobile Agent Systems," *Proceedings of IEEE MASCOTS*, College Park, Maryland, USA, pp. 314–321, October 1999.

[164] S. M. Yacoub, H. Xue, and H. H. Ammar, "Automating the Development of Pattern-Oriented Designs for Application Specific Software Systems," *Proceedings of IEEE ASSET '00*, Richardson, TX, USA, pp. 163–170, March 2000.

[165] G. Florijn, M. Meijers, and P. van Winsen, "Tool Support for Object-Oriented Patterns," *Proceedings of ECOOP '97*, Finland, pp. 472–495, June 1997.

[166] M. Schutze, J. P. Riegel, and G. Zimmermann, "PsiGene – A Pattern-Based Component Generator for Building Simulation," *Journal Theory and Practice of Object Systems (TAPOS)*, Vol. 5, No. 2, pp. 83–95, April 1999.

[167] T. D. Meijler, S. Demeyer, and R. Engel, "Making Design Patterns Explicit in FACE," *Proceedings of ESEC/FSE '97*, Zurich, Switzerland, pp. 94–110, September 1997.

[168] A. H. Eden, A. Yehudai, and J. Gil, "Precise Specification and Automatic Application of Design Patterns," *Proceedings of IEEE ASE '97*, Lake Tahoe, LA, USA, pp. 143–152, November 1997.