Notice

Cabletron Systems reserves the right to make changes in specifications and other information contained in this document without prior notice. The reader should in all cases consult Cabletron Systems to determine whether any such changes have been made.

The hardware, firmware, or software described in this manual is subject to change without notice.

IN NO EVENT SHALL CABLETRON SYSTEMS BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS MANUAL OR THE INFORMATION CONTAINED IN IT, EVEN IF CABLETRON SYSTEMS HAS BEEN ADVISED OF, KNOWN, OR SHOULD HAVE KNOWN, THE POSSIBILITY OF SUCH DAMAGES.

Virus Disclaimer

Cabletron has tested its software with current virus checking technologies. However, because no anti-virus system is 100% reliable, we strongly caution you to write protect and then verify that the Licensed Software, prior to installing it, is virus-free with an anti-virus system in which you have confidence.

Cabletron Systems makes no representations or warranties to the effect that the Licensed Software is virus-free.

Copyright © July 1996, by Cabletron Systems, Inc. All rights reserved.

Printed in the United States of America.

Order Number: 9030647 E6

Cabletron Systems, Inc.
P.O. Box 5005
Rochester, NH 03866-5005

SPECTRUM, the SPECTRUM IMT/VNM logo, DCM, IMT, and VNM are registered trademarks, and SpectroGRAPH, SpectroSERVER, Inductive Modeling Technology, Device Communications Manager, and Virtual Network Machine are trademarks of Cabletron Systems, Inc.

C++ is a trademark of American Telephone and Telegraph, Inc.
UNIX is a trademark of UNIX System Laboratories, Inc.
OSF/Motif and Motif are trademarks of the Open Software Foundation, Inc.
X Window System is a trademark of X Consortium, Inc.
Ethernet is a trademark of Xerox Corporation.
Restricted Rights Notice

(Applicable to licenses to the United States Government only.)

1. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.


2. (a) This computer software is submitted with restricted rights. It may not be used, reproduced, or disclosed by the Government except as provided in paragraph (b) of this Notice or as otherwise expressly stated in the contract.

   (b) This computer software may be:

   (1) Used or copied for use in or with the computer or computers for which it was acquired, including use at any Government installation to which such computer or computers may be transferred;

   (2) Used or copied for use in a backup computer if any computer for which it was acquired is inoperative;

   (3) Reproduced for safekeeping (archives) or backup purposes;

   (4) Modified, adapted, or combined with other computer software, provided that the modified, combined, or adapted portions of the derivative software incorporating restricted computer software are made subject to the same restricted rights;

   (5) Disclosed to and reproduced for use by support service contractors in accordance with subparagraphs (b) (1) through (4) of this clause, provided the Government makes such disclosure or reproduction subject to these restricted rights; and

   (6) Used or copied for use in or transferred to a replacement computer.

   (c) Notwithstanding the foregoing, if this computer software is published copyrighted computer software, it is licensed to the Government, without disclosure prohibitions, with the minimum rights set forth in paragraph (b) of this clause.

   (d) Any other rights or limitations regarding the use, duplication, or disclosure of this computer software are to be expressly stated in, or incorporated in, the contract.

   (e) This Notice shall be marked on any reproduction of this computer software, in whole or in part.
Preface

How to Use This Guide ........................................................................................................ vii
Related Reading .................................................................................................................. viii
Getting Help .................................................................................................................... ix

Chapter 1 Introduction

SPECTRUM Perspective .................................................................................................... 1-1
Some General Terms .......................................................................................................... 1-2

Chapter 2 Inductive Modeling Technology

The SPECTRUM Knowledge-Base .................................................................................... 2-1
The Knowledge-Base Management System ..................................................................... 2-3
  Building the SPECTRUM Knowledge-Base ............................................................... 2-4
  Communicating with Managed Nodes ........................................................................ 2-5
  Presentation of SPECTRUM Knowledge .................................................................... 2-5
  The Inferencing Mechanism ........................................................................................ 2-6

Chapter 3 SPECTRUM Terminology

Associations ........................................................................................................................ 3-1
Modeling a Network with Verbs and Nouns .................................................................... 3-2
Attributes and Inference Handlers ................................................................................... 3-4
Alerts, Events, and Alarms ............................................................................................... 3-6
Actions ................................................................................................................................ 3-7
Views, Icons, and Information Blocks ............................................................................ 3-7
The Landscape .................................................................................................................. 3-8
Handles .............................................................................................................................. 3-9
SPECTRUM Knowledge-Base Organization .................................................................... 3-9
  The Model Type Hierarchy ......................................................................................... 3-9
  The Semantic Data Model .......................................................................................... 3-11
## Chapter 4  SPECTRUM Architecture

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECTRUM Client-Server Architecture</td>
<td>4-1</td>
</tr>
<tr>
<td>SPECTRUM Overview</td>
<td>4-3</td>
</tr>
<tr>
<td>Model Type Editor</td>
<td>4-4</td>
</tr>
<tr>
<td>SpectroGRAPH</td>
<td>4-5</td>
</tr>
<tr>
<td>Views</td>
<td>4-5</td>
</tr>
<tr>
<td>Icons</td>
<td>4-6</td>
</tr>
<tr>
<td>The Rib Language</td>
<td>4-6</td>
</tr>
<tr>
<td>SpectroSERVER Interface</td>
<td>4-7</td>
</tr>
<tr>
<td>SpectroSERVER</td>
<td>4-7</td>
</tr>
<tr>
<td>VNM</td>
<td>4-8</td>
</tr>
<tr>
<td>Inference Handlers</td>
<td>4-9</td>
</tr>
<tr>
<td>Database</td>
<td>4-9</td>
</tr>
<tr>
<td>Modeling Catalog</td>
<td>4-10</td>
</tr>
<tr>
<td>Device Communication Manager (DCM)</td>
<td>4-10</td>
</tr>
<tr>
<td>Distributed SpectroSERVER (DSS)</td>
<td>4-11</td>
</tr>
<tr>
<td>Threads</td>
<td>4-12</td>
</tr>
<tr>
<td>Handles and Object Interfaces</td>
<td>4-13</td>
</tr>
<tr>
<td>Actions</td>
<td>4-15</td>
</tr>
<tr>
<td>Polling</td>
<td>4-16</td>
</tr>
<tr>
<td>Logging</td>
<td>4-16</td>
</tr>
<tr>
<td>Management Modules</td>
<td>4-17</td>
</tr>
</tbody>
</table>

## Glossary

## Index
Chapter 2  Inductive Modeling Technology

Figure 2-1. Knowledge-Base Interactions ................................................................. 2-3

Chapter 3  SPECTRUM Terminology

Figure 3-1. Inference Handler Interfaces ................................................................... 3-5
Figure 3-2. Model Type Derived from Multiple Base Model Types ......................... 3-10
Figure 3-3. Semantic Data Model ........................................................................... 3-11

Chapter 4  SPECTRUM Architecture

Figure 4-1. SPECTRUM Client-Server Relationship .................................................. 4-2
Figure 4-2. SPECTRUM Components .................................................................... 4-3
Figure 4-3. SpectroSERVER Components ................................................................. 4-8
Figure 4-4. Device Communication Manager ......................................................... 4-11
Figure 4-5. Distributed SpectroSERVER ................................................................. 4-12
Figure 4-6. SPECTRUM Management Module ..................................................... 4-17
The SPECTRUM Concepts Guide is the starting point for anyone using either the SPECTRUM Level I or Level II Developer’s Toolkit. The information provides a basic explanation of SPECTRUM functionality and shows how it is implemented as a management system. It is designed to help toolkit developers understand how SPECTRUM operates and assist them in building SPECTRUM extensions.

### How to Use This Guide

This guide is organized as follows:

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1 Introduction</td>
<td>Explains the reasoning behind the SPECTRUM approach to network management.</td>
</tr>
<tr>
<td>Chapter 2 Inductive Modeling Technology</td>
<td>Describes the Inductive Modeling Technology (IMT) that is the basis of SPECTRUM.</td>
</tr>
<tr>
<td>Chapter 3 SPECTRUM Terminology</td>
<td>Defines how each of the IMT concepts is defined as an aspect of SPECTRUM.</td>
</tr>
<tr>
<td>Chapter 4 SPECTRUM Architecture</td>
<td>Provides a detailed look at how the various components of SPECTRUM work.</td>
</tr>
<tr>
<td>Appendix A Glossary</td>
<td>Defines SPECTRUM terms.</td>
</tr>
</tbody>
</table>
Related Reading

The following Cabletron documents contain further information on the topics discussed in this guide:

- **SPECTRUM Basic Extension Guide**
- **SPECTRUM Extensions Integration Toolkit Developer’s Guide**
- **SPECTRUM GIB Editor Guide**
- **SPECTRUM Global Classes Reference**
- **SPECTRUM IIB Editor Guide**
- **SPECTRUM Inference Handler API Developer’s Guide**
- **SPECTRUM Knowledge-Base Guide**
- **SPECTRUM Model Type Editor Guide**
- **SPECTRUM MSAP/EPI Developer’s Guide**
- **SPECTRUM View API Developer’s Guide**
- **SPECTRUM VnmParmBlock Reference**
- **SpectroSERVER Application Programming Interface Developer’s Guide**
- **SpectroSERVER Application Programming Interface Reference**
- **SPECTRUM Report Generator User’s Guide**
Getting Help

For additional support for SPECTRUM products, or to make comments or suggestions regarding SPECTRUM or this manual, contact Cabletron Systems Technical Support via one of the following means:

<table>
<thead>
<tr>
<th>Location</th>
<th>Mail</th>
<th>FAX</th>
<th>Telephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>North America</td>
<td>Cabletron Systems, Inc. P. O. Box 5005 Rochester, NH 03866-5005 E-mail: <a href="mailto:spectrum-support@ctron.com">spectrum-support@ctron.com</a></td>
<td>603-337-3075</td>
<td>603-337-3500</td>
</tr>
<tr>
<td>Europe</td>
<td>Cabletron Systems, Ltd. Network House Newbury Business Park London Road, Newbury Berkshire, England RG13 2PZ E-mail: <a href="mailto:eurespec@ctron.com">eurespec@ctron.com</a></td>
<td>(*)-44-635-552062</td>
<td>(*)-44-635-580000</td>
</tr>
<tr>
<td>Pacific</td>
<td>Cabletron Systems, Inc. Allambie Grove Estate 25 French's Forest Road East French's Forest, NSW 2086 Sydney, Australia E-mail: <a href="mailto:spectrum-support@ctron.com">spectrum-support@ctron.com</a></td>
<td>(*)-61-2-950-5950</td>
<td>(*)-61-2-950-5900</td>
</tr>
<tr>
<td>Japan</td>
<td>Cabletron Systems, KK JTB Building 9F 164 Maranouchi Chiyoda-ku Tokyo 100 Japan E-mail: <a href="mailto:spectrum-support@ctron.com">spectrum-support@ctron.com</a></td>
<td>(*)-81-3-3240-1985</td>
<td>(*)-81-3-3240-1981</td>
</tr>
<tr>
<td>Singapore</td>
<td>Cabletron Systems, Inc. 85 Science Park Drive #03-03/04 The Cavendish Singapore 051 E-mail: <a href="mailto:spectrum-support@ctron.com">spectrum-support@ctron.com</a></td>
<td>(*)-65-7763382</td>
<td>(*)-65-7755355</td>
</tr>
<tr>
<td>Germany</td>
<td>Cabletron Systems GmbH Dreieich Park Im Gefierth 13d 63303 Dreieich Frankfurt, Germany E-mail: <a href="mailto:spec-germany@ctron.com">spec-germany@ctron.com</a></td>
<td>(*)-49-6103/991-229</td>
<td>(*)-49-6103/991-269</td>
</tr>
</tbody>
</table>

*International Operator Code

Questions About SPECTRUM Documentation?
Send your questions, comments or suggestions regarding SPECTRUM documentation to the Technical Communications Department directly via the following internet address:

spectrum-techdocs@ctron.com
Getting Help
Chapter 1

Introduction

This document presents the concepts related to the SPECTRUM approach to an integrated management system. This document is a prerequisite for both Level I and Level II Toolkit development; it explains both the underlying technology of SPECTRUM and the terminology used in the rest of the SPECTRUM documents. Knowing these concepts and terms, SPECTRUM developers will have the background to extend SPECTRUM functionality, and to make extensions to SPECTRUM at the appropriate points. Chapters 2-4 describe the underlying concepts of SPECTRUM. The rest of this chapter discusses the recent history of the computer network industry that led to the development of SPECTRUM, and defines some networking terms as they apply to SPECTRUM.

SPECTRUM Perspective

When the SPECTRUM project was first started, the deficiencies of network management systems were well known. With the phenomenal growth of local area networks (LANs) and the advent of internets, computer networks had become so large and complex that they were almost unmanageable. The diversity of devices, vendors, and network administrative bodies exposed the shortcomings of network management technology specific to a particular vendor. As a result, standard protocols for network management emerged, such as the Simple Network Management Protocol (SNMP) and the Common Management Information Protocol (CMIP). However, vendors continued to produce devices that were managed by their own proprietary protocols. These developments in the industry created a need for a network management solution that would support both industry-standard and proprietary network management protocols.

SPECTRUM was designed to provide such a solution. SPECTRUM addresses many of the problems of network management.

- It detects and controls the state of the network for both logical and physical configurations.
- It manages the performance of the network by controlling and analyzing the throughput and error rate.
• It performs fault management by detecting, isolating, and controlling abnormal network behavior.

• It collects and processes data related to resource consumption in the network.

In addition, SPECTRUM’s management capabilities can be extended in many directions. For example, SPECTRUM can be extended to manage new types of devices on a network, and to use the new management protocols that manage these devices.

SPECTRUM can also be extended to solve problems in domains other than network management, such as facilities management and factory automation. For example, SPECTRUM extensions can manage a climate controller or a factory robot. SPECTRUM extensions can also manage software products such as an operating system, a database management system, a financial planning software package, or even another network management system. All these extensions can be easily integrated into the existing SPECTRUM framework. Once these extensions are integrated into SPECTRUM, the management information gathered by SPECTRUM from one domain is combined with information gathered from other domains to provide a complete picture of the entire enterprise.

Some General Terms

Network management circles often use the same terms to mean different things. In SPECTRUM, these terms are used in a somewhat broader sense.

The SPECTRUM definition of a network management system consists of three components:

• Several managed nodes, each of which has access to a management agent

• A network management station

• A network management protocol, which is used by the station and the agents to exchange management information about the nodes

A Managed Node is either a host system such as a computer or a printer, a gateway, or a media device such as a bridge or a hub. In SPECTRUM, this definition can also include a control instrument such as a thermostat or a humidifier, a software product such as an operating system or an accounting package, or a machine on an automated factory floor, such as a robot.

A Network Management Station is a host system that runs the network management applications and protocol. In SPECTRUM, a network management station can be expanded to manage intelligent facilities and automated factories.
A **Network Management Protocol** is the means by which the management station and the managed nodes exchange management information. SPECTRUM can exchange management information via a number of industry-standard protocols. If a managed node can only exchange management information via a proprietary protocol, SPECTRUM provides an interface to convert the information into a form that can be interpreted by SPECTRUM.

A **Managed Object** is a variable of a managed node. This variable contains one piece of information about the node. Each node can have several objects. The management station monitors and controls the node by reading and changing the values of these objects.

A **Management Agent** is a process that exchanges the managed node’s information with the management station. The agent can either be resident on the node, or it can be a proxy residing elsewhere that acts on behalf of the node.
Chapter 2

Inductive Modeling Technology

The basis of SPECTRUM is the Inductive Modeling Technology (IMT). The technology is heavily influenced by concepts and techniques developed in various fields of artificial intelligence, such as linguistics, knowledge representation, and problem-solving. Although it cannot be categorized as either a semantic network or a model-based system, IMT integrates the concepts and technologies of both. The first section of this chapter describes the SPECTRUM knowledge-base with a simple example using the theory behind IMT. The rest of this chapter elaborates on the example and shows how IMT allows SPECTRUM to manage a network.

The SPECTRUM Knowledge-Base

The core of the IMT is a knowledge-base that represents everything that SPECTRUM can model. The knowledge-base is made up of concepts and relationships between concepts. A concept is made up of two components: its declarative knowledge (what it is) and its procedural knowledge (how it behaves). The specific instances of concepts and relationships that comprise a given knowledge-base should reflect the application for which SPECTRUM is being used.

The following example is a very simple illustration that shows how the concepts, relationships, and behaviors can reflect an application. This example shows how models of a Port, a Hub, and a Room interact with each other.

The first step is to define the declarative and procedural knowledge for the concepts.

Concepts:

PORT, HUB, ROOM

Declarative Knowledge:

PORT: Serial Port, Parallel Port.
HUB: Ports, Cables.
ROOM: Walls, Floors, Windows.

Procedural Knowledge:
PORT: How to react to a DEVICE, How to manage a HUB, How to react to a LOCATION.
HUB: How to react to a PORT, How to react to a LOCATION.
ROOM: How to react to a DEVICE, How to react to a COMPONENT.

Then some relationships should be defined between the concepts:

Relationships:
CONNECTS_TO, CONTAINS, IS_ADJACENT_TO

Next, specific instances of the concepts and relationships just defined must be added to the knowledge-base.

Instances of Concepts:

Port 12: Instance of PORT.
Engineering Hub: Instance of HUB.
Demo Room: Instance of ROOM.

Instances of Relationships:

Engineering Hub CONNECTS_TO Port 12
QA LAN IS_ADJACENT_TO QA Hub
Research Building CONTAINS Demo Room

This is a simple illustration with only a few concepts and relationships, and a limited amount of declarative and procedural knowledge. A knowledge-base used for complex applications such as network management or factory automation would contain many more concepts and relationships, and a vast amount of declarative and procedural knowledge.
The Knowledge-Base Management System

All interaction with the knowledge-base takes place through the Knowledge-Base Management System. The outside world interacts with the knowledge-base in three distinct ways. Figure 2-1 shows these interactions.

Figure 2-1. Knowledge-Base Interactions

1. The knowledge-base has to be built up or populated with concepts, relationships, and specific instances of those concepts and relationships. Some of these instances should represent the “managed nodes” that SPECTRUM communicates with.

2. After the knowledge-base is built, SPECTRUM begins to exchange management information with the managed nodes, to monitor them and to control them.

3. Users can interact with the knowledge-base to retrieve information, and to send management commands to the managed nodes.
The next three sections discuss these steps in detail. The knowledge-base management system also provides a powerful inferencing mechanism that builds upon the notion of concepts and their relationships. This mechanism is explained in the section entitled The Inferencing Mechanism on page 2-6.

Building the SPECTRUM Knowledge-Base

The basic SPECTRUM package includes a comprehensive knowledge-base that contains many concepts and their relationships. This knowledge-base also includes ways of presenting these concepts and relationships to a user, and protocols for managing nodes.

More concepts and relationships, along with their presentation formats, can be introduced into the knowledge-base by adding various SPECTRUM management modules. A user can also introduce more instances of existing concepts and their relationships into the knowledge-base. Using the previous example, if Demo Room and Engineering are added as instances of the existing ROOM and HUB concepts, the relationship Demo Room CONTAINS Engineering Hub can be established.

New concepts and their relationships can also be added to the knowledge-base, if the new concepts are derived from existing concepts. New concepts and relationships can be classified as either simple or advanced.

Simple concepts are those that can be derived from existing concepts by adding only declarative knowledge (knowledge about what it is). To create a concept of an AUI PORT, an administrator can derive from the existing PORT concept and add the declarative knowledge that this concept of a Port has an AUI Connector.

If new procedural knowledge (knowledge of how to do something) has to be added to existing concepts, the new derived concepts are considered advanced concepts. Suppose the concept of a CHIPCOM_HUB is derived from HUB. The declarative knowledge that identifies it as a CHIPCOM_HUB must be added to correctly identify the concept. However, the procedural knowledge of how to react to a PORT is based on a HUB concept that responds to commands from Spectrum. Therefore, the procedural knowledge for responding inherited from the HUB concept is insufficient, and procedural knowledge of dependent functioning inherited from the PORT must be added to the CHIPCOM_HUB concept. Adding this procedural knowledge now makes the CHIPCOM_HUB concept an advanced concept.

Defining new concepts and their relationships in the knowledge-base is considered to be extending the capabilities of SPECTRUM. These extensions can be categorized as either Level I (simple) or Level II (advanced) extensions. An example of Level I extension is adding simple concepts and relationships by using the Model Type Editor and the mmbuild script, as detailed in the SPECTRUM Basic Extension Guide. SPECTRUM developers can add Level II extensions, for example, by defining concepts and relationships using
Communicating with Managed Nodes

SPECTRUM can exchange management information with specific managed nodes once instances of those nodes have been added to the knowledge-base. Only nodes that have access to a management agent can be managed. SPECTRUM can gather information from these managed nodes and send commands to manage them.

The SPECTRUM knowledge-base understands some management protocols, such as the Simple Network Management Protocol (SNMP) and the Internet Control Management Protocol (ICMP Ping). SPECTRUM can manage any nodes that use these protocols. If a managed node does not use the protocols included with the basic SPECTRUM package, a protocol converter can be developed using the SPECTRUM External Protocol API. This protocol converter converts the management messages used by the managed node into messages that are understood by SPECTRUM's External Protocol Interface (EPI). To communicate with SNMP managed nodes, the Management Module for the SNMP Protocol Module must be installed with SPECTRUM. To communicate with EPI managed nodes, the Management Module for the EPI Protocol Module must be installed with SPECTRUM.

Presentation of SPECTRUM Knowledge

SPECTRUM uses a graphical format to present its knowledge to a user. In this format, users can look at the knowledge from different perspectives or views. The instances of the concepts in the SPECTRUM knowledge-base are shown as icons in each view. A view can contain one or more icons of instances that are related in some way. Each icon shows different characteristics of the instance and may change its appearance depending on the condition of the instance it is representing. In the previous example, the knowledge-base may be presented to a user from the perspective of its location, showing the building in a location where Port 12, HUB_CSI_TRMM, and DEMO ROOM are at a given moment, with each of them being represented on the screen by different icons. Each icon would reflect the condition of the instance it is representing. For instance, when Port 12 is connected to the HUB_CSI_TRMM, the icons for both may show a neutral grey color. If Port 12 loses connection, the icon representing Port 12 may change to green, and the icon representing HUB_CSI_TRMM may flash red.

A number of views and icons come with the basic SPECTRUM package. Installing more SPECTRUM management modules can add other types of views. New icons can be created to represent instances of new concepts that
may have been added. Also, existing views can be modified to show the new icons, or new views can be created to present SPECTRUM knowledge from an entirely new perspective.

The Inferencing Mechanism

The knowledge-base contains representations of the managed nodes with which it can exchange management information. When changes occur in these managed nodes SPECTRUM is notified, and its inference mechanism ensures that these changes reach the appropriate destinations in the knowledge-base. If these changes trigger more changes in other parts of the knowledge-base, the inference mechanism also propagates these changes.

Using the previous example, assume that procedural knowledge has been attached to the concept of PORT such that every time a HUB IS_ATTACHED_TO a PORT, the PORT FLASHES RED. Assume also that procedural knowledge has been attached to the concept of ROOM so that whenever a PORT FLASHES RED, the ROOM IS_NOT_ACCEPTABLE for the HUB. Therefore, when the HUB_CSI_TRMM is connected to the Port 12, the inference mechanism causes the Port 12 to overheat, and ROOM to be too warm for the Port 12.

The example of a knowledge-base containing concepts such as PORT, HUB, and ROOM is a very general one. An application based on such a knowledge-base can take full advantage of the IMT for modeling, inferring and reasoning. SPECTRUM was developed as a management tool, especially for network management. The main purpose of the example is to highlight the potential of SPECTRUM for modeling complex systems in addition to managing networks, automated manufacturing facilities and transaction processing systems.
Chapter 3

SPECTRUM Terminology

Concepts, relationships, instances of concepts, and relationships between instances that exist in the knowledge-base are referred to by special terms in the SPECTRUM documentation. This section explains these and other related terms.

Associations

The knowledge-base of SPECTRUM consists of a large number of related concepts. Linguistically, these concepts can be thought of as nouns or noun phrases, and the relationships between the concepts as verbs or verb phrases. In SPECTRUM terminology, the nouns are called model types, and the verbs are called relations.

Stringing noun and verb phrases together forms a sentence. For a sentence to be meaningful, it has to meet three criteria:

• It must be in the format (subject) noun + verb + (object) noun
• It must make logical sense -- one cannot use any verb to link any two nouns
• It must reflect reality

Noun and verb phrases in a knowledge-base must be very carefully defined so that when they are combined into a sentence, the sentence meets the first criterion.

The SPECTRUM notion of rules ensures that the second criterion is met. Rules can be defined on a verb to limit the nouns that the verb can link. Again, the rules need to be defined carefully so that the restrictions they impose on the verbs make logical sense. Typically, each verb is governed by several rules. Note that these are not semantic rules, or rules of a rule-based AI system.

Newly-created instances of model types and relations must meet the criterion that a sentence reflects reality. If these instances do not reflect reality, the information gathered and presented by SPECTRUM will also not reflect reality.
Model types, relations, and rules form the basic building blocks of the SPECTRUM knowledge-base. Model types and relations are abstractions of real-world objects, and rules are logical restrictions on the way these abstractions can be linked to each other.

Specific instances of the nouns, or particular concepts, are called models. When a model is linked to another model by a certain relation (within the specified rules of that relation), the two models form an association.

A model, therefore, is an instantiation of a model type, a specific object of a class of objects. An association is an instantiation of a relation. Only those relations that follow the rules can be instantiated, and only those model types that are defined as instantiable can be instantiated by SpectroGRAPH. The process of making a logical sentence that the SPECTRUM knowledge-base can use consists of two steps: creating models, and creating associations between models. The models that are created must reflect objects that exist in the real world; the associations that are created must reflect the relationships between the objects in the real world.

Modeling a Network with Verbs and Nouns

Consider a language in which the following nouns and verbs are defined as the model types and relations that reflect the objects that make up the parts of a network. Further, let’s say rules are defined to impose a logical sense on the way the nouns and verbs can be strung together. Rules are defined for a relation, and consist of two model types -- a left model type and a right model type. This left-right order in rules signifies the subject (the left model type), the verb (the relation), and the object (the right model type) that can form a logical sentence.

Nouns:
“building”, “room”, “network”, “LAN”, “printer”, “workstation”

Verbs:
“contains”, “collects”

Rules:
“contains” [ building, room ], [ room, workstation ]
“collects” [ LAN, printer ], [ LAN, workstation ], [ network, LAN ]

To make statements that make logical sense in this language, the first two criteria have to be met. The following meet the first and second requirements and are realistic representations of a network:

“engineering building contains testing lab”
“testing lab contains Pat’s workstation”
“Engineering LAN collects Pat’s workstation”
“Engineering LAN collects LaserJet printer”
The following are invalid because they do not use the noun/verb/noun format, and therefore do not meet the first criteria:

“contains building collects”

“room LAN workstation”

The following meet the format requirement, but are either illogical or do not follow the defined rules:

“building contains workstation”

“LAN collects room”

“printer collects LAN”

Next, models that are instances of the actual concept being modeled can be created. Typical examples of models are: “The corporate office building”, “Roger’s office”, “The corporate network”, “The LAN on the first floor”, and “The PS-1 (printer)”.

We can now make statements by forming associations between the models, such as:

• “The corporate office building” contains “Roger’s office”
• “The LAN on the first floor” collects “Dave’s workstation”
• “The corporate network” collects “The LAN on the first floor”
• “Roger’s office” contains “Dave’s workstation”

The above statements may or may not be reflecting reality. The last statement probably does not reflect reality because Roger’s office probably contains his own workstation, rather than Dave’s workstation. However, if this statement is introduced into the SPECTRUM knowledge-base, SPECTRUM would continue to operate as though Roger’s office contains Dave’s workstation.

Similarly, if the following rule was also defined in the language,

“contains”: [ workstation, building ].

SPECTRUM would permit associations to be built that meet the rule, even though it is obviously illogical. For example, the statement, “Dave’s workstation contains the Corporate office building” would be permitted because it meets the second criterion. Thus, all newly defined rules must make logical sense, and models and associations between models must reflect reality.

Relations are defined within SPECTRUM as either one-to-many or many-to-many. The Contains relation is an example of a one-to-many relation. A workstation can only be in one building, not several buildings. In contrast, Connects is a many-to-many relation. A workstation can connect with multiple devices, such as a server and a printer. New relations should also be defined as one-to-many or many-to-many.
Attributes and Inference Handlers

When a concept is introduced into the SPECTRUM knowledge-base, knowledge about its contents and its behavior must be defined. The contents form the declarative knowledge of the concept -- knowledge of what the concept is. The behavior forms the procedural knowledge of the concept, or the knowledge of how the concept does something.

In SPECTRUM, the declarative knowledge of a model type is defined by the attributes of the model type. Each attribute describes a part of the model type. For example, some of the attributes of the model type “building” may be the name, the address, and a telephone number. Some of the attributes of the model type “workstation” may be the serial number, its ethernet address, and its MIPS rating. The values of all the attributes of a model collectively define its internal state. This distinguishes two models of the same model type from each other.

The behavior of a model type is defined by one or more intelligence circuit. Each intelligence circuit is a collection of inference handlers. Each inference handler can perform some task. The task may be as simple as changing the value of an attribute, or it may be as complex as discovering all the devices on a network. An inference handler may perform a generic task like calculating an average, or it may perform a task specific to a model type, such as creating models of boards in a hub. When a number of inference handlers work together to perform a more complex task, they form an intelligence circuit.
Inference handlers define the behavior of the model type for a specific set of conditions. They can define the behavior of a model type if a model is created or destroyed, or if it is activated (a model is activated when it establishes communication with the managed node that it is modeling). They can define the behavior if the value of an attribute of a model changes, or if an event is generated by the model (see the section on Alerts, Events and Alarms). They can define the behavior if a model forms a new association with another model or is removed from an existing association. Finally, they can define how certain actions (see the section on Actions) are to be handled. Figure 3-1 shows the interfaces to an inference handler.

**Figure 3-1. Inference Handler Interfaces**

Inference handlers define how a model type reacts to external conditions. Inference handlers also execute on behalf of the models of the model type. Thus, when the external condition of two models of a model type changes in a similar way, the reaction of both the models is similar. However, the internal state of the model (i.e., values of its attributes) is also an input to an inference handler. The internal state of one model is different from that of the other. Thus, even though the external condition is the same and the inference handlers react in a similar way on behalf of each model, the end result of the reaction by the two models can be different.
For example, inference handlers are defined for the Hub model type to create models of the boards of the hub whenever a new hub model is created. In addition to the external stimulus (creation of the hub model), the number and type of each of the models that are to be created is input to the inference handlers. The Hub model type has attributes for the number and type of boards; each hub model has different values for these attributes. Thus when a hub model is created, the same inference handlers create a different (but appropriate) number and type of models of boards.

Alerts, Events, and Alarms

Once a managed node’s models are created in the knowledge-base and the node has access to a management agent, SPECTRUM can begin to exchange management information with a managed node. Usually, SPECTRUM initiates the communication to gather information from or send commands to the managed nodes. However, some managed nodes can initiate the communication.

When a managed node spontaneously initiates communication with SPECTRUM, the managed node is said to have generated an alert. (The SNMP term for an alert is trap.) For example, the printer PS-1 may generate an alert when it is turned on or off, or runs out of paper. The list of alerts generated by a managed node is built into the SPECTRUM knowledge-base when the model type of the node is defined. Additional alerts can be defined for new model types built with the Level I or Level II toolkit.

When a model in SPECTRUM detects something unusual, it may generate an event. An event indicates that something significant has happened, usually in a model. All events that are generated are logged in the event log. An alert reported by a managed node causes an event to occur in the model representing the node. However, events are not caused only by alerts. An event can occur when a model is created or destroyed, when one of its attributes crosses some threshold, etc. For example, when the model for the printer PS-1 is created, an event is generated. When PS-1 generates an alert when it is turned on or off or runs out of paper, more events are generated. The basic SPECTRUM package defines some generic events that are common to all model types, or are significant to SPECTRUM. More events can be introduced into the SPECTRUM knowledge-base when new model types are defined. If the new model types define new alerts, the alerts can be mapped into new events.

An event can indicate that an abnormal condition exists, in which case the user must be informed of the abnormal condition immediately. Abnormal conditions are reported to a user with an alarm, by flashing screens, ringing bells, or some other means. The model type designer has to decide which new events should be mapped to alarms. For example, the model type designer for the printer may decide that the event created by the alert “PS-1 out of paper” should not generate an alarm, but the event created by the alert “PS-1 turned
off” should generate an alarm. In addition, if the alert “PS-1 out of paper” is being generated by the printer every five minutes, the model type designer can create a new event, “printer running out of paper too often”, and generate an alarm every time this event is generated. Both the events within the basic SPECTRUM package and events that are generated from alerts sent to new model types can be configured as alarms, according to the model type designer’s requirements. When new events are defined as part of new model types, some may have to be mapped into new alarms.

Actions

SPECTRUM defines a small set of operations that can be performed on a model, such as reading or writing an attribute. SPECTRUM also provides a powerful mechanism called an action to expand this basic operations set. An action is any operation that is not part of the basic set of operations defined by SPECTRUM for use with a model. A model type designer can define a new action and then must provide the inference handler(s) to implement it.

For example, the designer of the model type “workstation” can define an action called “EXECUTE_PROGRAM” that is carried out through inference handlers (also provided by the model type designer) by all models of the model type “workstation”. When invoked with the proper parameters, this action triggers the inference handlers that initiate communication with the workstation and starts a program on the workstation. Since actions can be implemented only by new inference handlers, the Inference Handler Developer’s Toolkit is needed to define new, meaningful actions. The actions can then be invoked through the SpectroSERVER API of the SPECTRUM Developer’s Toolkit.

Most operations can be performed without defining actions. For example, to start a program on a workstation without defining a new action, set up the workstation so that the program is started up whenever some environment variable is set. Then set this environment variable on the workstation via SPECTRUM. (See the section entitled Actions on page 4-12 for more details about this example.)

Views, Icons, and Information Blocks

The SPECTRUM knowledge-base can be seen by users from different perspectives. Each perspective is called a view. Each view presents different information, each in its own presentation style. For example, one view shows how managed nodes are logically connected to each other. Another view shows the physical locations of the managed nodes. Some of the views are hierarchical and provide a capability for the user to progressively look at more detailed sections of the view. For example, a user can look at the contents of a building and then zoom in on particular rooms of interest. Other views, called
generic views, provide gauges, graphs, and other visual aids to monitor the activity of a managed node. The user can run a number of views simultaneously; each view functions independently of the other views.

Each view uses a different style to present its information. The view that shows logical connections of managed nodes, for example, has small lines or pipes, whereas the view that shows the physical locations of managed nodes has flags and boxes. Each view can contain a number of items or icons. These icons represent information from the SPECTRUM knowledge-base. Some icons represent entire models and appear as flags of locations or small pictures of managed nodes or other basic shapes for networks or cables. Each such icon is a live picture of that model. Some of the large icons can contain live gauges or other status fields that change color or move in real time as the data in the knowledge-base changes. Icons that represent models are actually collections of smaller icons placed upon a base icon.

Information about the configuration of views and icons, such as the arrangement of the smaller icons on base icons, and what icons should be displayed in a view, is specified in UNIX text files called information blocks. The basic SPECTRUM package contains a number of information blocks that specify this information. When new base icons are needed to represent new types of models, or new icons need to be displayed in a view, these can be added to the appropriate information block files.

The Landscape

The landscape is composed of the models, associations, attribute values, alarms, events, and statistics specific to any one virtual network machine (SpectroSERVER) in a single network. A network can consist of several SpectroSERVERs (subnets); however, each SpectroSERVER can only contain a single landscape. Each landscape contained in a network is unique and must be identified by a unique landscape handle (ID). This becomes of particular importance in networks broken down into subnets where landscape information is to be shared from one SpectroSERVER to another. If instances of landscapes having the same handle exist, the resulting ID contention will make communication between SpectroSERVERs impossible.
Handles

Each model type, relation, model, attribute, and landscape is uniquely identified in SPECTRUM by its handle. These handles are hexadecimal numbers generated by SPECTRUM. All communication within SPECTRUM uses handles to match data with its correct destination. These handles are described more fully in the section entitled “Handles and Object Interfaces” in Chapter 4.

SPECTRUM Knowledge-Base Organization

As mentioned previously, a knowledge-base comes with the basic SPECTRUM package, consisting of some model types, relations and rules. The SPECTRUM management modules add more model types, relations, and rules to the SPECTRUM knowledge-base. This knowledge is organized as a hierarchy of model types as well as a semantic network of model types and relations (a semantic data model). The following sections give a brief conceptual overview of the model type hierarchy and the semantic data model. The SPECTRUM Knowledge-Base Guide describes the actual hierarchy of model types and the network of model types and relations.

The Model Type Hierarchy

The model types are built in a hierarchical fashion, with the more general model types being built first, and the more specific model types being derived from the more general ones. In building the model type hierarchy, the notion of inheritance is an important one. There are two aspects to this notion of inheritance.

First, when a model type is derived from a general one, it inherits both the attributes and the intelligence circuits from the base model type. The derived model type also participates in all the relations in which the base model type participates. If the derived model type does not describe a new concept completely, more attributes and/or intelligence circuits can be added to it. Thus, the derived model type is a more specific type than its base.

Second, a model type can be derived from any number of other model types (multiple inheritance). Thus, a derived model type is made up of the attributes and intelligence circuits of one or more base model types. It also participates in all the relations in which all its base model types participate.
The hierarchy implicitly establishes two relations between the model types: the IS_A relation and HAS relation. When a model type is derived from another, an implicit relationship is established between the two, which is called the IS_A relation. That is, the derived model type IS_A specific type of the base model type. Multiple inheritance is a means of showing that a derived model type IS_A combination of a number of base model types. Often, it may appear that a concept IS_A combination of a number of other concepts. However, a more detailed analysis shows that the concept actually IS_A specific type of one concept, and HAS several additional capabilities (the HAS relation). Derivation is the only way to implicitly establish the IS_A relation as well as the HAS relation between model types. However, the model type hierarchy does not distinguish between the two implicit relations and model types in the hierarchy.

In the current model type hierarchy, SPECTRUM uses multiple inheritance in a number of cases, but rarely to establish the IS_A relation. When a model type is derived from a number of base model types, it usually IS_A (more specific) type of one of the base model types, and HAS the other base model types. It is recommended that SPECTRUM customers, too, use multiple inheritance mainly to establish the HAS relation. For example, a capability may be needed in a number of model types that are not related by the IS_A relation. An easy way to accomplish this is to develop a model type having this capability, and then to derive the model types that need the capability from this one. Figure 3-2 shows how a derived model type is formed from multiple base model types.

Figure 3-2. Model Type Derived from Multiple Base Model Types

In this figure, Router IS_A Device, Room IS_A Location, and WorkStation IS_A Device; and Hub HAS the DataRelay capability and Router HAS the DataRelay capability; WorkStation IS_A Device, but HAS no DataRelay capability.
A derived model type inherits attributes and intelligence circuits from its base model types in a specific order to prevent the derived model type from inheriting an attribute or an intelligence circuit multiple times. The order also determines the initial value of an attribute. This order is explained in detail in the SPECTRUM Inference Handler Application Programming Interface Developer's Guide.

**The Semantic Data Model**

The model type hierarchy shows the relations IS_A and HAS between two model types. However, it cannot show more complicated relations between model types. To show how model types participate in different relations with other model types, a semantic network of model types and relations is needed. In SPECTRUM, this is called the semantic data model (SDM). The semantic data model shows model types at the highest level in the hierarchy that can participate in each relation. It implies that model types derived from the ones shown in the data model also participate in the relations in the same way. However, in some cases there may be rules that restrict the participation of a derived model type in the relation shown in the model.

Figure 3-3 shows a simple semantic data model, with the model types of Figure 3-2 participating in the relation, “Contains”. The figure shows that models of model type Device can participate in the Contains relation with models of model type Location.

Also, models of Location can participate in the Contains relation with other models of Location. Figure 3-3 also implies that a model of a type derived from the Device model type can participate in a relation with a model of a type derived from the Location model type. For example, it implies that Room models can Contain Hub models, or WorkStation models, or Router models. Similarly, Building models can Contain Hub models, Router models, or WorkStation models. It also implies that Building models can Contain Room models, and Room models can Contain Building models.

Since the last case does not make logical sense, the following rule restricts the participation of model types derived from Location, in the Contains relation:

“Contains”: [ Building, Room ]
The rule is not depicted in the semantic data model. For more details about the SDM, see the SPECTRUM Knowledge-Base Guide.
SPECTRUM Architecture

This chapter provides an overview of the SPECTRUM architecture. It illustrates SPECTRUM's implementation of the Inductive Modeling Technology (IMT) and describes the major components of SPECTRUM.

SPECTRUM Client-Server Architecture

SPECTRUM is an example of a client-server architecture, a system structure based on the relationship between a process that provides services (the server) and an application process that uses the services (the client). The server process knows how to service the request and hides this information from the client. The client knows how to interface with the server but requires no knowledge of how to provide the requested service. For example, the server process is often a hardware interface that relieves the client from knowing how to communicate with different managed nodes. A server can communicate with multiple clients, regardless of whether they reside on the same computer as the server. Figure 4-1 shows the SPECTRUM client-server relationship.
The SPECTRUM client process is SpectroGRAPH, which requests information about managed nodes. SpectroSERVER is the server process that responds to the request, providing the necessary information. Information can be retrieved from a database or directly from the managed node. The client doesn’t need to know how the information is retrieved, but only how to request information from SpectroSERVER. Also, clients can ask SpectroSERVER to send unsolicited messages about the changing status of a managed node. Other SpectroSERVER clients include the SPECTRUM Command Line Interface, SPECTRUM AR Gateway, and SPECTRUM Data Gateway. All clients communicate with SpectroSERVER through the SpectroSERVER Application Programming Interface (SSAPI).

There are two modes provided for communication through the SSAPI. Synchronous mode causes the client program to pause execution until the request has been serviced and the response returned, at which point the program continues. Asynchronous mode works with a “callback” feature. The client initiates the request and returns to continue execution of whatever it was doing. When SpectroSERVER has the reply ready, the client receives a callback to indicate that the response is ready to be retrieved. These modes of communicating with SpectroSERVER are explained in the SpectroSERVER Application Programming Interface Developer's Guide.
SPECTRUM Overview

The SPECTRUM knowledge-base management system broadly encompasses the concepts of IMT in its implementation, including:

- Declarative knowledge in the form of model types, attributes, rules, relations, models and associations.
- Ability to communicate with managed nodes
- Presentation of and interaction with information
- Procedural knowledge in the form of inference handlers

Figure 4-2 illustrates the SPECTRUM knowledge-base management system. The large triangle represents the SpectroSERVER and includes SpectroSERVER access layer, represented by the shaded area, and the knowledge-base, represented by the white sections within the smaller triangle. The figure depicts information being added to the knowledge-base through the MTE and client applications. Also shown is a client application interfacing with SpectroSERVER, and managed nodes communicating with the Device Communications Manager (DCM).

Figure 4-2. SPECTRUM Components
SpectroSERVER provides the interfaces to access most of the information in the knowledge-base. The knowledge-base consists of all the defined concepts, both stored in the database and contained within the procedural knowledge of inference handlers in SpectroSERVER, and presentation formats, implemented using external files, accessed by the client. (It may be possible to store the presentation information in external files in the database in future releases. This information is still considered part of the knowledge-base, although it is not shown in this diagram.)

Inference handlers are actually part of SpectroSERVER, but contain the procedural knowledge of the knowledge-base. The declarative knowledge of model types and relations are added directly into the knowledge-base through the MTE. Clients such as SpectroGRAPH add models and associations through SpectroSERVER, which in turn accesses the database to store the information in the knowledge-base.

Model Type Editor

The Model Type Editor (MTE) is the primary tool that defines the concepts, rules, relationships, and declarative knowledge in the SPECTRUM knowledge-base. The MTE is a stand-alone executable with its own user interface. MTE output is stored in the database for access by SpectroSERVER. The MTE and SpectroSERVER cannot execute at the same time, since they both require exclusive use of the database (see the section entitled “Database Manager”).

The MTE can create and modify SPECTRUM model types (concepts), attributes, their descriptions and default values (declarative knowledge), relations (logical connections) and rules (restrictions). Existing model types can be customized to meet specific needs, and new model types can be derived from existing model types.

An attribute defined by the MTE for a model type describes a part of the model type, such as Model Type Name or Total Errors. For managed nodes of the same model type, the attribute value reflects the internal state of the managed node. Attributes can be defined as being pollable or not pollable, depending on whether the managed node is queried periodically to update the attribute value. The attribute Model Type Name is an example of a static, non-pollable attribute, whereas Total Errors is pollable. The MTE can also define polling intervals (how frequently the managed node is queried to update pollable attributes). Be careful when you define a polling interval; too-frequent polling can cause unnecessary network traffic. SpectroSERVER handles SPECTRUM polling (see the section entitled Logging on page 4-13).
Using the MTE to define the knowledge-base is an important step in SPECTRUM and requires careful planning to determine the hierarchies of model types being defined. A model type that is derived from an existing model type inherits the attributes and SPECTRUM intelligence of the base model type. The SPECTRUM Model Type Editor Guide describes how to use the MTE to define the knowledge-base.

**SpectroGRAPH**

SpectroGRAPH is the graphical user interface to the SPECTRUM system and a client of SPECTRUM. An application developed using the X Window System™ and OSF/Motif™, SpectroGRAPH provides a means to navigate through the system, create, delete and update models of managed nodes, examine information about managed nodes, and manipulate the information provided by the system.

Models created using SpectroGRAPH are instances of the model types existing in the database. Several different models that represent different managed nodes can share the same attributes if they are of the same model type. Models and the information from the knowledge-base are displayed from various perspectives, or views, and models in a view are represented using icons.

**Views**

SpectroGRAPH provides a number of different views including Location, Network Topology, Device, and Generic Informational views. An external file, CsViewControl, contains information that defines how views are to be presented, and how to package each view for display. The file defines and describes all of the existing SpectroGRAPH view types and can be used to define new view types. External files in the CsPib directory control which images can represent models in Location, Network Topology, and Device Views. External files in the CsGib directory control the contents of the generic views. The SPECTRUM GIB Editor enables users to directly manipulate generic views through SpectroGRAPH (see the SPECTRUM GIB Editor Guide).

If developers want to write applications that are integrated with and displayed by SpectroGRAPH, the SPECTRUM View API provides such an interface. Refer to the SPECTRUM View API Developer's Guide for more information about this process. Also refer to the SPECTRUM Basic Extension Guide for more information about SPECTRUM views and the CsViewControl file.
Icons

Each view is comprised of icons and other displayable objects. An icon is simply a graphic or picture representation displayed on a screen. When a view is created, it creates all the necessary icons to be displayed in that view. Icons can be very simple and consist of a single image that represents entire models in the knowledge-base, or made up of several layers of icons. Device icons usually consist of the initially displayed icon, called a “primary-icon”, and several smaller icons, called “sub-icons”. Sub-icons can be double-clicked by the user to provide additional information through menu selections, bringing up new views, or executing scripts. For more information on icons, refer to the SPECTRUM IIB Editor Guide.

The Rib Language

The Rib Language is a simple programming language that lets you specify the format and contents of a report. You first create a Rib file with a text editor or the Rib Editor. The lexical analyzer checks if the Rib file is lexically correct (i.e., the information is presented in the right order, the Rib blocks are correctly enclosed within braces, etc.). The lexical analyzer also creates the data structure required by the Rib Parser. The Rib Parser then parses the output of the lexical analysis to check that the Rib file is syntactically correct. A Rib file may be rejected if a fatal error is encountered at either the lexical analysis or parsing stage. The Rib file then needs to be corrected and passed through the Lexical Analyzer and the Rib Parser again. If a Rib file is both lexically and syntactically correct, the Rib Parser generates the data structure required by the Report Generator to generate the formatted report.

The Rib Editor helps you write Rib files that are lexically and syntactically correct. The Rib Editor enforces most of the rules of the Rib Language by only allowing the valid options depending on the stage of editing you are in. The remaining rules are enforced by the Rib Parser, which is called by the Rib Editor each time you try to save or preview the Rib file. For this reason, it is highly recommended that you create new Rib files only through Rib Editor, not your general purpose text editor.

It is possible to create a Rib file (through Rib Editor) that is lexically and syntactically correct but still is improperly formatted. The Rib Editor Preview allows you to view the format of the report without having to actually generate a report. It is strongly recommended that you use this facility and make the necessary corrections in the Rib file to correctly format the report. Reports created using the Rib Editor can be displayed in tabular or graphical formats. For more information on the Rib file, refer to the SPECTRUM Report Generator User’s Guide.
SpectroSERVER Interface

All information about SpectroGRAPH models is obtained through requests to the SpectroSERVER API. The SpectroSERVER Application Program Interface Developer's Guide describes how other client applications can make similar requests of SpectroSERVER. SpectroGRAPH also receives unsolicited responses from SpectroSERVER as events, statistics, alarms, or information pertaining to the changing state of a managed node, as explained in Chapter 3, SPECTRUM Terminology. When SpectroGRAPH receives such information, the corresponding icons are updated to reflect the changed status of the managed nodes.

Alarm information is displayed using the Alarm View, while events are displayed in the Event View. The type of information displayed in these views is described in the SPECTRUM System User's Guide. For an explanation of how events, alarms, statistics, and alerts are processed, see the SPECTRUM Basic Extension Guide.

SpectroSERVER

SpectroSERVER serves as the Knowledge-Base Management System and is the center of the SPECTRUM system. The components within SpectroSERVER, depicted in Figure 4-3, provide required information for the client requests.
The Virtual Network Machine (VNM) is the controlling part of SpectroSERVER. It receives requests from the SpectroSERVER API and calls on the appropriate component to service the request. Information requested through the API can already be in memory, or it may require database or managed node access. If the information is not in memory, the VNM passes the request to either the Device Communications Manager or the database manager.

During the evolution of the SPECTRUM product, SpectroSERVER and VNM have been used interchangeably so often as to become synonymous; however, for explicitness, we have differentiated the two terms in this section of the guide.
Inference Handlers

The VNM contains procedural knowledge, or SPECTRUM intelligence, in the form of inference handlers that describe how a model type reacts to changes in its environment. Inference handlers usually handle the behavior of a newly defined model type. Inference handlers can cause an unsolicited response to be returned to the client application in response to detected changes in the model.

Inference handlers are attached to a model type. When an inference handler is invoked, it works on behalf of the individual model of the model type to which it is attached. However, two models of the same model type might still react differently to the same external stimuli depending on the external stimulus, the reaction to that stimulus through the inference handler, as well as the internal state of the model.

Several inference handlers may be required to define how a model type should react. This group of inference handlers is referred to as an intelligence circuit for the given model type. The intelligence circuits available with existing model types are described in the SPECTRUM Knowledge-Base Guide. The SPECTRUM Inference Handler API Developer's Guide describes how to write new intelligence circuits.

Database

The database provides storage for specific device configurations, statistics and events, and a modeling catalog (model types and relations). SpectroSERVER contains an object-oriented database, written before commercial object-oriented databases were readily available. To help create the object-oriented database, Cabletron chose a network database management system called db_VISTA™, by Raima Corporation. In a network database, relationships between record types are explicitly defined and directly maintained through sets. The concept of sets allows the definition of a one-to-many relationship between two record types. The linked lists of pointers to record locations of the set members and owners result in a network of interconnected records. The primary benefit provided with a network database is better performance.

Model types, rules, relations and attributes are defined through the MTE. Models and associations are defined using SpectroGRAPH or some other client. Information retained from one execution of SpectroSERVER to the next is also stored in the database. SpectroGRAPH and other client applications do not access the database directly but by requesting information through the SpectroSERVER API calls. This eliminates the need to understand the implementation of the database management software in SpectroSERVER.
Modeling Catalog

When you model multiple landscapes using Distributed SpectroSERVER, the database for each landscape must contain identical modeling catalogs. This means that all the model types that exist in one landscape's modeling catalog must also exist in the modeling catalog of every other landscape. When a particular model type is not available in your local database, models of that model type appear as Undefined Icons. If you install new management modules in one landscape, you must install the same management module on every landscape.

Device Communication Manager (DCM)

The Device Communication Manager is the interface between the VNM and the managed nodes. The DCM includes various clients that communicate with specific managed nodes via its protocol. There is one client for each protocol supported.

Several protocol interfaces, such as SNMP, are available for SPECTRUM. When a SpectroSERVER request needs to communicate with the managed node, the VNM sends the request on to the appropriate protocol interface client in the DCM. The DCM in turn passes the request on to the managed node.

SPECTRUM's External Protocol Interface API (EPI API) enables SPECTRUM to communicate with a managed node that does not communicate with one of the supported protocols. (Currently, EPI, ICMP ping (echo), and SNMP protocols are supported.) The Management Station Access Provider (MSAP) accesses SPECTRUM through the External Protocol Interface API. An MSAP is a software task or process that can be running in any environment -- the same environment as SPECTRUM, the same environment as the managed node, or in a completely different environment. The MSAP can use any mechanism to communicate with the managed node, and provide the managed node with an interface to SPECTRUM.

Figure 4-4 illustrates DCM communication with devices. Refer to the SPECTRUM MSAP/EPI Developer's Guide to determine how to write an MSAP that can communicate between SPECTRUM and a managed node.
Distributed SpectroSERVER (DSS)

Distributed SpectroSERVER (DSS) is a powerful modeling feature that uses the concept of landscapes. Refer to the section entitled “The Landscape” in Chapter 3 for more information about landscapes. DSS can improve SPECTRUM performance when managing a large network by distributing the load introduced by management traffic and allowing you to delegate network management functions to remote workstations. Using DSS, you can create a model of your network as a single landscape or logically partition it into subnets and create multiple landscapes, each with its own local SpectroSERVER.

Figure 4-5 shows multiple landscapes using DSS. When you create a network model with multiple landscapes, it is possible for a single SpectroGRAPH (or other application) to access information from more than one SpectroSERVER at the same time. Each remote landscape is identified by a unique landscape handle and represented by a landscape icon in the SpectroGRAPH to which it is connected. The landscape icon shows a rollup summary of alarm information from the associated SpectroSERVER and allows you to navigate from one landscape to another.
Threads

Since SpectroSERVER must handle requests from many clients and also access the disk and network, it includes a multi-threaded architecture that provides less overhead than separate UNIX processes. A scheduling subsystem called MOOT (Modular Object Oriented Threads) provides context-switching between threads.

Only one thread executes at any given time; the MOOT maintains a queue of other threads waiting for execution. SpectroSERVER creates some threads at start-up that terminate only when the SpectroSERVER terminates. The SpectroSERVER creates other threads dynamically, then terminates them when they are no longer needed.

For example, each time a client connects with SpectroSERVER or makes a request through the API, a new thread is started. The SPECTRUM Inference Handler API Developer's Guide describes in detail the multi-threaded environment and its effect on an inference handler.
Handles and Object Interfaces

Handles are unique numbers generated by SPECTRUM. Handles identify an object and provide an interface to the functions that access the object. Developers use handles instead of pointers to an object.

There are several different forms of handles used in the SPECTRUM system for each type of object being identified. The number assigned to a given object identifies that object, regardless of whether the object is being accessed in the client application or through an inference handler within SpectroSERVER. The handle provides an interface to classes defined by the type of object, which in turn provides access to the object. The classes contain different information, depending on whether the access to the object is at the client level or from an inference handler.

The following handle classes can access an object of type Model Type:

<table>
<thead>
<tr>
<th>Type of Object</th>
<th>Model Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Identification</td>
<td>CsMTypeHandle</td>
</tr>
<tr>
<td>Inference Handlers</td>
<td>CsVnmMTypeHandle</td>
</tr>
<tr>
<td>Client - Asynchronous</td>
<td>CsUIModelTypeHandle</td>
</tr>
<tr>
<td>Client - Synchronous</td>
<td>CsSYModelTypeHandle</td>
</tr>
</tbody>
</table>

The basic name of the handle identifying this type of object is CsMTypeHandle; this is also the identifying number. Clients identify the same object as CsUIModelTypeHandle if using the Asynchronous SpectroSERVER API, or CsSYModelTypeHandle if using the Synchronous SpectroSERVER API. Both classes provide the client application with access to methods used in dealing with this type of object. Both classes use the number defined in CsMTypeHandle to identify the object.

Inference handlers identify this same object using CsVnmMTypeHandle, a class that provides an inference handler in SpectroSERVER with access to methods used in dealing with this type of object. This class also uses the same number defined in CsMTypeHandle to identify the object at this level.
The following are the different forms of handles for other types of objects:

<table>
<thead>
<tr>
<th>Type of Object</th>
<th>Model Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Identification</td>
<td>CsModelHandle</td>
</tr>
<tr>
<td>Inference Handlers</td>
<td>CsVnmModelHandle</td>
</tr>
<tr>
<td>Client - Asynchronous</td>
<td>CsUIModelHandle</td>
</tr>
<tr>
<td>Client - Synchronous</td>
<td>CsSYModelHandle</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type of Object</th>
<th>Landscape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Identification</td>
<td>CsLandScapeHandle</td>
</tr>
<tr>
<td>Inference Handlers</td>
<td>CsVnmLscepHandle</td>
</tr>
<tr>
<td>Client - Asynchronous</td>
<td>CsUILandScapeHandle</td>
</tr>
<tr>
<td>Client - Synchronous</td>
<td>CsSYLandScapeHandle</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type of Object</th>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Identification</td>
<td>CsRelHandle</td>
</tr>
<tr>
<td>Inference Handlers</td>
<td>CsVnmRelationHandle</td>
</tr>
<tr>
<td>Client - Asynchronous</td>
<td>CsUIRelationHandle</td>
</tr>
<tr>
<td>Client - Synchronous</td>
<td>CsSYRelationHandle</td>
</tr>
</tbody>
</table>

Attributes also have a unique identifier called an Attribute ID. However, developers access attributes through their model instead of accessing them directly.

Classes are provided for developer access to the Alarm Manager, Event Log Manager and Statistics Log Manager. Since there is only one object in the system for each of these, they carry an implicit identifier.

The following shows the classes used to access these objects through the SpectroSERVER API:

<table>
<thead>
<tr>
<th>Type of Object</th>
<th>Alarm</th>
<th>Statistics</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client - Asynchronous</td>
<td>CsUIAlarm</td>
<td>CsUIStat</td>
<td>CsUIEvent</td>
</tr>
<tr>
<td>Client - Synchronous</td>
<td>CsSYAlarm</td>
<td>CsSYStat</td>
<td>CsSYEvent</td>
</tr>
</tbody>
</table>
The SpectroSERVER Application Program Interface Developer's Guide describes how to use these classes.

Actions

SPECTRUM includes a small set of actions, operations that can be performed on a model. Actions are identified by their action code. SPECTRUM developers can add actions that are not part of the basic set of actions using the Level II Developer Kit. To implement the example from Chapter 3 of adding the EXECUTE_PROGRAM action, a developer would perform the following steps:

1. Identify the action that is required. Actions and their corresponding action codes are defined in a header file. The action EXECUTE_PROGRAM and an associated action code must be added to this file. Refer to the SPECTRUM VnmParmBlock Reference for information about creating action code files.

2. Develop the code necessary to implement the action as an inference handler in SpectroSERVER. The code must register the EXECUTE_PROGRAM action, connect to the workstation, and start execution of the program. The inference handler gets linked into SpectroSERVER. The Inference Handler API Developer's Guide explains what is needed to write such an inference handler.

3. Invoke the EXECUTE_PROGRAM action in the application. Use the appropriate synchronous or asynchronous SpectroSERVER API to communicate with the inference handler.
Polling

The information clients receive through the SpectroSERVER API includes a list of a model's attributes whose current values are requested. The VNM determines if the requested attribute has been defined through the MTE as external, to be obtained from the managed node, or internal, stored either in memory or the database. Some external attributes are defined as polled, meaning that the VNM polls the managed nodes on a regular basis, based on the value of the polling interval which is defined for the model. If the requested attribute is external but not defined as a polled attribute, the VNM obtains the value from the managed node at the time of the request.

The MTE can also define polling groups for attributes. The VNM groups the attributes as one request to be polled at the same time. Grouping attributes reduces network traffic, which can be very active if there are many pollable attributes. The default polling interval is one request per model every 60 seconds for each pollable attribute. Increasing the polling interval to every 30 seconds without grouping the attributes can generate an unacceptable amount of network traffic. The Device Communication Manager (DCM) handles communication with the managed node.

Logging

Attributes can also be defined through the MTE as being logged, meaning that their values are written to the SPECTRUM Statistics Log at regular intervals. The frequency with which values are logged is based on both the polling interval and the logging ratio defined for the attribute. For example, if a polling interval of 60 is defined for the attribute, and the logging ratio is set to 10, the attribute value is logged to the statistics file every tenth poll, or every 600 seconds.
Management Modules

Management modules can be designed by SPECTRUM developers to address the problem of managing new nodes in SPECTRUM. Figure 4-6 shows the components of the management module.

To develop a management module, first use the MTE to create new model types for the new objects in the database. The SPECTRUM Knowledge-Base Guide explains how to determine model type derivation for the new model types and what intelligence is inherited automatically when deriving from other model types. The SPECTRUM Model Type Editor Guide describes how to use the MTE to create new model types.

Next, determine how SpectroGRAPH should display the model of the managed node. By using the SPECTRUM GIB Editor and following the instructions in the SPECTRUM Basic Extension Guide and SPECTRUM GIB Editor Guide, developers create generic views to present data reflecting the statistics of the managed node and define icons that represent the managed node in SpectroGRAPH Views. Once you have done this, SpectroGRAPH creates a model for the new managed node. At this point, the intelligence circuit in SpectroSERVER for that model works together with the rest of the system to allow SPECTRUM to manage the new node.

New intelligence can be created for the new managed node with the SPECTRUM Inference Handler Developer’s Toolkit. With this toolkit, code can be developed to implement the inference handlers and intelligence circuits to enable SpectroSERVER to react to the activity of the managed node. The
SPECTRUM Inference Handler API Developer's Guide describes how to write inference handlers.
action
Any operation that is not part of the basic set of operations defined by SPECTRUM for use with a model.

agent
See network management agent.

API
See application programming interface.

application programming interface (API)
A set of routines used to make calls to another software package.

association
A link formed between two models by a relation.

asynchronous call
A call to a method that begins, but does not necessarily complete, the requested operation before allowing the program to continue execution. At some point in time, the requested operation completes and notifies the program. In the meantime, the program and the requested operation can both proceed at the same time. See also synchronous call.

attributes
The declarative knowledge in SPECTRUM that defines what a concept is. Attributes are defined using the Model Type Editor.

client
The application process in a client-server architecture. See also client-server architecture.

client-server architecture
A system design based on the relationship between a process that provides services, referred to as the server, and an application process that uses the services, referred to as the client.

csi
Cabletron Systems image, a file containing image data. Used as part of a file name (.csi).
database
A collection of interrelated data organized to facilitate efficient and accurate inquiry and update. See also database management system and db_VISTA™.

database management system
A software package that organizes and maintains a database. See also db_VISTA™.

db_VISTA™
The database management system, by Raima Corporation, used by SPECTRUM.

DCM
See Device Communications Manager.

declarative knowledge
Information defining what a concept is in SPECTRUM.

device
A network element of some kind. In SPECTRUM, the word device and managed node are often used interchangeably, since the most common implementation of SPECTRUM concepts is currently network management.

Device Communications Manager (DCM)
A multi-protocol communication engine in the SpectroSERVER that handles communication with all network devices, regardless of their protocol. The DCM translates SpectroSERVER requests into protocol understood by the individual devices.

Distributed SpectroSERVER (DSS)
A modeling feature that uses the concept of landscapes to improve SPECTRUM performance when managing a large network by distributing the load introduced by management traffic and allowing you to delegate network management functions to remote workstations.

Edit Mode
Allows editing the current view in SPECTRUM. Selecting the Edit Mode displays the File and Edit options in the menu bar.

EPI
See External Protocol Interface.

event
A significant message from the SpectroSERVER.

External Protocol Interface (EPI)
An interface that provides communication with multiple protocols, but hides communication details from SPECTRUM and MSAP.

Generic Information Block (GIB)
Parameters for controlling generic screen views in SPECTRUM. See also GIB View.
generic view
  See GIB View.

GIB
  See Generic Information Block.

GIB File
  Contains the parameter templates used to display a Generic View.

GIB View
  SPECTRUM views, referred to as Generic Views, that use templates to
determine what kind of information to present to a user, and the format to
use in presenting it. These templates are stored in a GIB file. A GIB view
displays a model's configuration, diagnostic, and performance information.

icon
  A graphic or picture representation displayed on a screen.

Icon Information Block (IIB)
  Used in SPECTRUM to describe how an icon is to be displayed.

IIB
  See Icon Information Block.

Inductive Modeling Technology (IMT)
  Cabletron’s set of artificial intelligence techniques that allow a network of
arbitrary complexity to be modeled such that every element of the
network is given intelligence.

inference handler
  Part of the procedural knowledge in SPECTRUM that defines how a
model type reacts.

instantiate
  In Object Oriented Design, creating a particular occurrence of something.

intelligence circuit
  A collection of inference handlers that defines the behavior of a model
type.

knowledge-base
  Everything that can be modeled and managed by SPECTRUM, including
concepts, relationships, declarative knowledge and procedural knowledge.

knowledge-base management system
  The software products used to define and manage the information in the
knowledge-base.

landscape
  All data specific to any one virtual network machine (VNM) in a single
network.
managed node
An item or device whose status is being monitored and controlled. In SPECTRUM the terms device and managed node are often used interchangeably, since the most common implementation of SPECTRUM concepts is currently network management. The more general concept of managed node involves any item being monitored, and does not necessarily have to refer to a network device.

managed object
A variable on a managed node containing one piece of information about the node. Each node may have several objects.

management agent
An implementation of a management protocol which exchanges the managed node’s information with the management station.

Management Station Access Provider (MSAP)
In SPECTRUM, a software task that provides an object with access to a management station.

model
Collection of information that forms a specific occurrence of some basic defined type. In SPECTRUM, models are instances of model types.

model type
A template that describes the attributes, actions, and associations related to a network entity in SPECTRUM.

Model Type Editor (MTE)
The primary tool that defines the concepts, relationships, rules, and declarative knowledge in the SPECTRUM knowledge-base.

Modular Object Oriented Threads (MOOT)
A task control manager used in the SpectroSERVER, developed at Cabletron.

MOOT
See Modular Object Oriented Thread.

Motif Window Manager (Mwm)
The window manager shipped with OSF/Motif™, which provides a unique “look and feel” to windows developed with that software.

MSAP
See Management Station Access Provider.

MTE
See Model Type Editor.

Navigation Mode
Moving from one view to the next in SPECTRUM. The menu bar contains the File and View options when in Navigation Mode. Navigation Mode does not allow editing of a view.
**network management agent**
An implementation of a management protocol that exchanges the managed node’s information with the management station.

**network management protocol**
The means by which the management station and the managed nodes exchange information.

**network management station**
The host system or workstation that is running the network management applications and protocol.

**network management system**
In SPECTRUM, the three components necessary to manage a complex network: several managed nodes, a network management station, and a network management protocol.

**Object-Oriented Design (OOD)**
A design encompassing the process of breaking a system into parts, each of which represents some class or object from the problem domain, and applied by viewing the world as a collection of objects that cooperate with one another to achieve some desired functionality. Typically includes a notation for depicting both logical and physical as well as static and dynamic models of the system under design.

**OID**
An identifier for a managed object.

**OOD**
See Object Oriented Design.

**OSF**
See Open Software Foundation.

**Open Software Foundation (OSF)**
A consortium of industry leaders that have united to direct, contribute to, and fund the continued development of the X Window System as an industry standard windowing system.

**Perspective Information Block (PIB)**
Information that maps model types to icon images in IIB files in SPECTRUM.

**PIB**
See Perspective Information Block.

**pollable attributes**
Attributes for which the VNM regularly queries the managed node to obtain current values. Attributes are defined as pollable or non-pollable through the Model Type Editor.

**procedural knowledge**
In SPECTRUM, information defining how a concept behaves or reacts to environmental changes.
**protocol**
A set of rules used by computers to communicate with each other.

**relation**
Information describing the semantic connection models have with each other.

**Report Information Block (RIB)**
An external file used for defining report formats.

**RIB**
See Report Information Block.

**rules**
Restrictions placed on relations in the knowledge-base.

**server**
A process that provides services in response to a client request in a client-server architecture. See also client-server architecture.

**SpectroGRAPH**
Graphical user interface software in SPECTRUM that provides a means for the user to view, edit and interact with the information provided by the system through the SpectroSERVER. A client in the client-server relationship with the SpectroSERVER.

**SpectroSERVER**
Software in SPECTRUM that controls communication with the database and the devices on the network, and acts as an interface between applications, such as the User Interface, and the information provided by the database and the devices. Major components within the SpectroSERVER include the Device Communications Manager and the Virtual Network Machine.

**SpectroSERVER API**
Provides a means for client applications to access information in the SpectroSERVER. See also application programming interface.

**SPECTRUM**
Cabletron’s protocol-independent advanced management platform for managing networks and networked systems. SPECTRUM is an X Windows™, OSF/Motif™ UNIX-based system.

**synchronous call**
A call to a method that performs the entire requested action before a program can continue running. The program continues only after the completion of the called method. See also asynchronous call.

**timing interval**
The frequency with which the current view updates displayed model attribute information. The default is one request per model every five seconds for some number of attributes. This can be modified by the user for a generic view.
UI (User Interface)
See SpectroGRAPH.

User Interface (UI)
See SpectroGRAPH.

view
One of many representations of the network landscape.

VNM
See Virtual Network Machine.

Virtual Network Machine
Within the SpectroSERVER, the software level that provides access to data regardless of where the data is stored. It can be stored in the database, the VNM’s memory, or any of the devices in the network. The VNM also embodies the SPECTRUM intelligence, known as the Inductive Modeling Technology.

window
A region on the display created by a client.

window manager
A client that allows the user to move, resize, circulate, and iconify windows on a display. The window manager used largely determines the “look and feel” of the X system on a particular system.

X
Abbreviation for X Window System.

X Window System
Network-based graphical windowing system.
A
action 3-5, 3-7, Glossary-1
advanced concepts 2-4
agent Glossary-1
Alarms 3-6
alert 3-6
Alerts 3-6
API
   see Application Programming Interface
Application Programming Interface (API) Glossary-1
architecture 4-1
association 3-2, Glossary-1
Associations 3-1
asynchronous call Glossary-1
asynchronous mode 4-2
attribute 3-4, Glossary-1
   pollable 4-4
attribute logging 4-16
Attributes 3-4

C
callback 4-2
client 4-1, Glossary-1
Client-Server Architecture 4-1
client-server architecture Glossary-1
CMIP 1-1
concepts 2-1
   advanced 2-4
   simple 2-4
csi Glossary-1

D
database Glossary-2
database management system Glossary-2
database manager, SpectroSERVER 4-9
db_VISTA 4-9, Glossary-2
DCM
   see Device Communications Manager
declarative knowledge 2-1, Glossary-2
device Glossary-2
Device Communications Manager (DCM) 4-17, Glossary-2

E
Edit Mode Glossary-2
EPI
   see External Protocol Interface
event 3-6, Glossary-2
Events 3-6
External Protocol Interface (EPI) 2-5, Glossary-2

G
General Terms 1-2
   managed nodes 1-2
   network management protocol 1-2
   network management station 1-2
Generic Information Block (GIB) Glossary-2
GIB File Glossary-3
GIB View Glossary-3

H
handle 3-9, 4-13
hierarchy, model type 3-9

I
icon 3-8, Glossary-3
Icon Information Block (IIB) Glossary-3
IIB
   see Icon Information Block
IMT
   see Inductive Modeling Technology
Inductive Modeling Technology (IMT) 2-1, Glossary-3
inference handler 3-4, 4-9, Glossary-3
Inference Handler Interfaces 3-5
Inference Handlers 3-4
Index (continued)
inference mechanism 2-6
Inferencing Mechanism 2-6
information block 3-8
intelligence circuit 3-4, Glossary-3

K
knowledge-base 2-1, Glossary-3
building 2-4
organization 3-9
Knowledge-Base Management System 2-3
knowledge-base management system 2-3, Glossary-3

L
landscape 3-8, Glossary-3, Glossary-4
logging
attribute 4-16

M
managed node 1-2
Managed Nodes 2-5
managed object 1-3, Glossary-4
management agent 1-3, Glossary-4
management module 4-17
Management Station Access Provider (MSAP) Glossary-4
model 3-2, Glossary-4
model type 3-1, Glossary-4
Model Type Editor (MTE) 4-4, Glossary-4
model type hierarchy 3-9
Modeling 3-2
Modular Object Oriented Threads (MOOT) 4-12, Glossary-4
MOOT
see Modular Object Oriented Threads
Motif Window Manager (Mwm) Glossary-4
MSAP
see Management Station Access Provider
MTE
see Model Type Editor
Multiple Inheritance 3-9

N
navigation mode Glossary-4
network management agent Glossary-5
network management protocol 1-2, 1-3, Glossary-5
network management station 1-2, Glossary-5
network management system Glossary-5
Notice i

O
object interface 4-13
Object-Oriented Design (OOD) Glossary-5
OOD
see Object-Oriented Design
Open Software Foundation (OSF) Glossary-5
OSF
see Open Software Foundation

P
Perspective Information Block (PIB) Glossary-5
PIB
see Perspective Information Block
pollable attributes 4-4, Glossary-5
polling groups 4-16
polling interval 4-16
procedural knowledge 2-1, Glossary-5
protocol Glossary-6

R
relation 3-1, Glossary-6
relationships 2-2
Report Information Block (RIB) Glossary-6
Restricted Rights Notice ii
RIB
see Report Information Block
Rib Language 4-6
rules 3-1, Glossary-6
S
SDM
    see semantic data model
semantic data model 3-11
server 4-1, Glossary-6
simple concepts 2-4
Simple Network Management Protocol (SNMP) 2-5
SNMP 1-1
    see Simple Network Management Protocol
SpectroGRAPH 4-2, 4-5, Glossary-6
    icons 4-6
    interface to SpectroSERVER 4-7
    views 4-5
SpectroSERVER 4-2, 4-7, Glossary-6
    actions 4-15
    database manager 4-9
    inference handlers 4-9
    threads 4-12
SPECTRUM Perspective 1-1
synchronous call Glossary-6
synchronous mode 4-2

T
Terminology 3-1
threads, SpectroSERVER 4-12
timing interval Glossary-6
Trademarks i
trap 3-6

V
view 3-7, Glossary-7
Virtual Network Machine (VNM) Glossary-7
Virus Disclaimer i
VNM
    see Virtual Network Machine

W
window Glossary-7
window manager Glossary-7

X
X Window System Glossary-7