IEEE Std 1226.3-1998

IEEE ABBET™

IEEE Standard for Software Interface for Resource Management for A Broad-Based Environment for Test (ABBET)

IEEE Standards Coordinating Committee 20

Sponsored by the
IEEE Standards Coordinating Committee 20 on
Information Technology

2 June 1998 *SH94610*

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

IEEE ABBET™

IEEE Standard for Software Interface for Resource Management for A Broad-Based Environment for Test (ABBET)

Sponsor

IEEE Standards Coordinating Committee 20

Approved 19 March 1998

IEEE-SA Standards Board

Abstract: The services needed to access and manage descriptive information about resources in an automatic test system (ATS) are covered. This information includes data about the automatic test equipment (ATE) instruments, switching, and the test subject adapter. This standard is a component of the ABBET set of standards.
Keywords: A Broad-Based Environment for Test (ABBET), automatic test equipment, automatic test system, resource management

---

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

Copyright © 1998 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1998. Printed in the United States of America.

ISBN 0-7381-0180-X

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

--`,,`-`-`,,`,,`,`,,`---

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS

Not for Resale

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; (508) 750-8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Introduction

[This introduction is not part of IEEE Std 1226.3-1998, IEEE Standard for Software Interface for Resource Management for A Broad-Based Environment for Test (ABBET).]

In the course of developing IEEE Std 1226-1993, A Broad-Based Environment for Test (ABBET), it was determined that multiple standards would be required to define the various ABBET functions and interfaces. These standards consist of the Overview and Architecture Standard, numbered 1226, and a set of related component standards, numbered 1226.1 through 1226.x. This component standard, numbered 1226.3, documents the software interfaces defined by ABBET to test equipment configuration information.

At the February, 1991, ABBET meeting, it was decided to document the automatic test equipment (ATE) system and instrument control in a single component standard. Deliberations subsequent to that meeting, however, revealed that combining the groups of interfaces into a single component standard would force implementations of the instrumentation interfaces to conform to a subset of that standard. Therefore, it was concluded that the interface groups should be documented in separate component standards, and that these standards would naturally divide at the interface between Test Equipment Description Language (TEDL) and Resource Description Language (RDL). At the Scottsdale, Arizona, meeting in May, 1991, this decision was changed, as it was decided to create three component standards for the ATE system layer: one for the resource management interfaces (referencing TEDL and RDL), one for the instrument interfaces (referencing SCPI, CIIL, etc.), and one for the instrument control bus interfaces (referencing IEEE 488, VXI, Ethernet, etc.).

This standard is intended to document the interfaces to services to ATE descriptive information that would allow bridging the gap between an ABBET ATLAS level test procedure and an ATE (referred to in this standard as “ATE interface services”).

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Participants

At the time this standard was completed, the ABBET Subcommittee of the IEEE Standards Coordinating Committee (SCC20) had the following membership:

Guy Adam
Fabio Angeli
James D. Baker
Jean-Yves Barbier
Darrell Bartz
Anne Bautista
Aldo Bernal
Michael Blair
Greg Bowman
Joanne Buckley
Toni Bunsen
Michael Carpenter
E. Elaine Cashar
Alexander Cherfas
Richard D. Churcher
Gregory W. Darnell
Bernard Dathy
Danny C. Davis
Hal Davis
Jose DeLaCruz
Bernd Dinklage
Bernard Dugas
Jerome Elisha
Keith Ellis
Michael T. Ellis
Paul Erath
Anthony Fitzpatrick
William Brit Frank
Michael Golas
Roger S. Goldberg
Arnold M. Greenspan
Jay Grove
Joel Guerra
Wes Hand
Arthur Hann
Gary L. Hardenburg
Robert G. Hayes
Leonard Haynes
Bill Heather
John E. Heiser
Shankar Hemmady
Ashley Hulme
Ulla Jacobson
Patrick Johnson
Andrew Joskowski
Al Karalius
Mark Kaufman
Lars-Olov Larsson
Hans Lecluyse
Robert Loretz, Jr.
Michael Lougue
Richard Maguire
Michael A. Malesich
Michael S. Martin
Gayle Matysek
Mark McCooey
Robert L. McGarvey
Harry McGuckin
Scott Miller
James M. Mills
Mukund U. Modi
William P. Neblett
David L. Nichols
Hans Obermeier
Leslie A. Orlidge
Ed Perkins
Jean Pouilly
N. Ramachandran
William E. Reeves
Bill Rodriquez
David Roggendorff
David Rollince
Eric Sacher
Howard Savage
Helmut Scheibenzuber
Rick Schippang
Michael L. Seavey
John W. Sheppard
Lee A. Shombert
William R. Simpson
Butch Sneade
Joseph J. Stanco
Jack Taylor
Kirk D. Thompson
Janette Tuttle
Jorg Urban
Kenneth Van
J. Richard Weger
Daniel H. Weiss
William A. Whitaker
Will W. Young
Jehuda Ziegler

In addition, the following people made significant contributions to this document:

Maria Bostic
Dez Cass
Jack Cross
Robert J. Curry
David L. Dawson
Ron DeBlanc
John Goddard
Chris Gorringe
T. Winston Griffin
Gary Hinkle
Glenn Laflin
John Leuthe
Rafael Martinez
Charles Morris
Ron Taylor
Mark J. Thullen

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

The following persons were on the balloting committee:

Guy Adam
James D. Baker
Jean-Yves Barbier
Jeffrey S. Barkley
Anne F. Bautista
Michael Blair
Greg Bowman
Alan L. Bridges
David M. Brooks
Theodor Calin
E. Elaine Cashar
Charles R. Childress
Richard D. Churcher
George A. Clarius
James E. Cottrell
Fred L. Cox
Robert J. Curry
Gregory W. Darnell
Bernard Dathy
Danny C. Davis
Somnath Deb
Bernd Dinklage
Horst D. Ey
William Brit Frank
Alfred S. Gilman
Arthur H. Greenberg
Arnold M. Greenspan
Gary L. Hardenburg
Robert G. Hayes
John E. Heiser
J. Luis Hernandez
Catherine Hogans
Ashley Hulme
Paul B. Knese
Scott A. Kotler
James L. Lewis
Richard J. Maguire
Michael A. Malesich
Michael S. Martin
Rafael Martinez
Gayle Matysek
Robert L. McGarvey
Thomas J. McGrath
Douglas W. McNabb
James M. Mills
Mukund U. Modi
Charles Morris
James Nagy
Richard E. Neese
David L. Nichols
Hans Obermeier
Leslie A. Orlidge
Richard J. Patrick
Dave Pawson
Dennis C. Peterson
Alain Plassart
Jean Pouilly
William E. Reeves
David Roggendorff
Eric Sacher
Howard Savage
Michael L. Seavey
John W. Sheppard
William R. Simpson
Philip T. Smith
Joseph J. Stanco
Robert Stasonis
Joseph L. Stautberg
Kirk D Thompson
Louis Y. Ungar
J. Richard Weger
Daniel H. Weiss
William A. Whitaker
Ronald T. Wolfe
Werner K. Wolz
Will W. Young
Robert W. Zimbardi

When the IEEE-SA Standards Board approved this standard on 19 March 1998, it had the following membership:

Richard J. Holleman, *Chair* **Donald N. Heirman,** *Vice Chair*
Judith Gorman, *Secretary*

Satish K. Aggarwal
Clyde R. Camp
James T. Carlo
Gary R. Engmann
Harold E. Epstein
Jay Forster*
Thomas F. Garrity
Ruben D. Garzon
James H. Gurney
Jim D. Isaak
Lowell G. Johnson
Robert Kennelly
E. G. "Al" Kiener
Joseph L. Koepfinger*
Stephen R. Lambert
Jim Logothetis
Donald C. Loughry
L. Bruce McClung
Louis-François Pau
Ronald C. Petersen
Gerald H. Peterson
John B. Posey
Gary S. Robinson
Hans E. Weinrich
Donald W. Zipse

*Member Emeritus

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Contents

1. Overview 1

1.1 Scope 1
1.2 Purpose 1
1.3 Application 1

2. References 2

3. Definitions and acronyms 2

3.1 Definitions 2
3.2 Acronyms 5

4. Description 5

4.1 Objectives 5
4.2 Overview 5
4.3 Relationship to ABBET architecture 6
4.4 Relationship to reference documents 9

5. Conformance 9

5.1 Minimum resource management interface functionality 9
5.2 Subsets 9
5.3 Extensions 9
5.4 Approach for implementation of subsets 10
5.5 Approach for implementation of extensions 10

6. Specification 11

6.1 Conventions 11
6.2 Virtual resource management services 13
6.3 Test equipment configuration management services 26
6.4 Real resource management services 32

Annex A (informative) Examples 47

Annex B (informative) Object-oriented analysis 62

Annex C (normative) Interface Definition Language (IDL) specification 71

Annex D (informative) Bibliography 86

Annex E (informative) Test foundation framework 87

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

IEEE ABBET™

IEEE Standard for Software Interface for Resource Management for A Broad-Based Environment for Test (ABBET)

1. Overview

The set of standards called “A Broad-Based Environment for Test (ABBET)” was developed to serve as a standards set for product testing. ABBET specifies a comprehensive environment for integrating design data, test strategies and requirements, test procedures, test results management, and test system implementations.

1.1 Scope

This standard is a component of the ABBET set of standards and covers the services needed to access and manage descriptive information about resources in an automatic test system (ATS). This information includes data about the automatic test equipment (ATE) instruments, switching, and the test subject to ATE adapter.

1.2 Purpose

This standard has a dual purpose. The first purpose is to define the interfaces to services that obtain descriptive capability and control information about an ATS. The second purpose is to define the interfaces to services that perform virtual and real resource management.

1.3 Application

This standard is primarily intended for use by ATS customers, ATE system developers and maintainers, and test program set (TPS) developers and maintainers. This standard will also be of interest to ABBET tool developers and test instrument developers.

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

2. References

This standard shall be used in conjunction with the following publications. When the following standards are superseded by an approved revision, the revision will be considered for incorporation into this standard.[1] Additional documents used in the development of this standard are listed in Annex D.

IEEE Std 100-1996, IEEE Standard Dictionary of Electrical and Electronic Terms.[2]

IEEE Std 716-1995, IEEE Standard Test Language for All Systems—Common/Abbreviated Test Language for All Systems (C/ATLAS).

IEEE Std 1226-1993, IEEE Trial-Use Standard for A Broad-Based Environment for Test (ABBET) Overview and Architecture.

OMG CORBA/IIOP Revision 2.1, The Common Object Request Broker: Architecture and Specification.[3]

IEEE P1226/D15.3, Draft Standard for A Broad-Based Environment for Test (ABBET) Overview and Architecture.[4]

IEEE P1226.4/D2.1, Draft Standard for Software Interface for Instrument Drivers for A Broad-Based Environment for Test (ABBET), 15 May 1996.

IEEE P1226.11/D4.1, Draft Standard for Test Resource Information for A Broad-Based Environment for Test (ABBET), 15 May 1998.

3. Definitions and acronyms

3.1 Definitions

IEEE Std 1226-1993 provides a complete glossary with definitions of all the terms used in the ABBET set of standards and guides. A clear understanding of the following terms with respect to an ATE system is particularly important in order to understand this ABBET component standard.

3.1.1 allocate: The process of identifying a resource for a particular task and making it unavailable to other users. The allocate process can occur during or prior to test execution. If a virtual resource is involved, the process of physical mapping is also included. *Contrast:* **deallocate.**

3.1.2 automatic test equipment (ATE): Test, measurement, diagnostic, and automation equipment that executes a sequence of manual or automated instructions that performs tests of functional or static parameters in order to evaluate the degree of performance degradation, or to perform fault isolation of unit malfunctions. Also included in ATE are the software packages required for the equipment to perform its functionality.

3.1.3 automatic test system (ATS): Includes the ATE as well as all support equipment, support software, test programs, and interface adapters.

[1]The capabilities of evolving reference standards will be integrated as they serve to improve the ABBET environment.

[2]IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

[3]This document is available from the Object Management Group (OMG), 492 Old Connecticut Path, Framingham, MA 01701, USA.

[4]Numbers preceded by P are IEEE authorized standards projects that were not approved by the IEEE-SA Standards Board at the time this publication went to press. For information about obtaining drafts, contact the IEEE.

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

3.1.4 behavior: A relationship between the inputs and the outputs of a product. As specified in ABBET terms, however, it is neither implementable nor testable until it has been specified in terms of signals that are observable or controllable. With respect to this standard, behavior is the observable and controllable signals that are used as inputs or outputs of a resource or path.

3.1.5 class: A description of a group of objects (or types) with similar properties, common services, common relationships, and common semantics.

3.1.6 composite resource: A specific type of real resource. It describes a collection of individual resources, or instruments, that provide some behavior when used collectively. Information known about the individual resources that make up the composite resource includes connectivity and the mapping of their individual resource ports onto the composite resource ports. *See also:* **real resource.**

3.1.7 connection: The virtual connection of a virtual resource that connects a test subject port to an external port of the test equipment configuration. A connection can be stated in ATLAS, for example, in the following manner: CNX HI P1-1. In this example, P1-1 is the test subject pin, and HI is the ATLAS reference to a port on the test equipment configuration.

3.1.8 connection set: A multiple list of connections used by a resource to connect multiple test subject ports to multiple output ports in the test equipment configuration. A connection set can be stated in ATLAS, for example, in the following manner: CNX HI P1-1 LO P1-2. In this example, the set is comprised of two connections, HI P1-1 and LO P1-2. Additionally, in rare cases, a list of connection sets can occur for resources that change connections during the implementation of their behavior, such as when checking for multiple ports. A set of connection sets can be shown in ATLAS by more than one set of HI, LO, or other test equipment configuration port designators linked to test subject ports.

3.1.9 deallocate: The process of releasing a resource from a particular task and making it available to other users. This is the reverse process of allocate. *Contrast:* **allocate.**

3.1.10 individual resource: A specific type of real resource that can be used by itself or be used jointly with other individual resources to make up a composite resource. The purpose of an individual resource is to generate or measure a class of signal. Information known about the individual resource includes its control interface and its current state. *See also:* **composite resource** and **real resource.**

3.1.11 instrument: An individual item that is synonymous with an individual resource. *Syn:* **individual resource.**

3.1.12 language-independent specification: The format for describing the services that are not tied to a specific computer language, syntax, or implied semantics.

3.1.13 object: A member (instance) of a class that encapsulates (i.e., hides the details of) state and operations.

3.1.14 path: Describes a contiguous routing through a test equipment configuration. Each path begins with a port and ends with a port. A path may include multiple switch connections and/or paths.

3.1.15 path set: Involves two or more paths that are needed to support a specific real resource. Many real resources have multiple ports that are active simultaneously and require multiple paths for the required connections. In addition, sets of path sets, or multiple path sets, may be required by a specific real resource.

3.1.16 physical mapping: The process of mapping a virtual resource to one or more real resources.

3.1.17 port: Describes the interface points on the test equipment configuration, the interface points on an individual resource, the interface points on a composite resource, the interface points on a virtual resource, and the interface points on a test subject.

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

3.1.18 real resource: An abstract supertype. The subtypes of real resources consist of individual resources or composite resources, both of which describe physical instruments. Unlike virtual resources, instances of the subtypes of real resources physically appear in a test equipment configuration.

3.1.19 resource: An abstract supertype in a test equipment configuration. A resource consists of virtual resources and real resources (even the test equipment configuration itself can be a resource). A resource is described by a set of ports and an optional set of behaviors.

3.1.20 service: A user-requested operation or runtime call whose behavior and interface is standardized.

3.1.21 signal: The observable behavior produced by or associated with a test procedure, resource, or path.

3.1.22 switch: A specific type of instrument or individual resource that is used for connecting or breaking an electric circuit.

3.1.23 switch connections: Routes through a switch that require a specific type of instrument or individual resource. A switch connection can be a part of a defined path.

3.1.24 test: A step in a test procedure that requires specific capabilities and produces an observation or response.

3.1.25 test equipment configuration: An integrated set of real resources. This composition includes all paths allowed by the interconnections of the real resources and all virtual resources that have been created using real resources. *See also:* **real resource.**

3.1.26 test foundation framework: A comprehensive set of object classes that supports the development of test programs and diagnostic data elements. The test foundation framework is defined in IEEE Std 1226-1993.

3.1.27 test method: Defines the algorithm and functionality of a test object.

3.1.28 test object: An object that represents a test of a specific behavior.

3.1.29 test procedure: Defines the implementation of a test method.

3.1.30 test program: Contains the coded sequences of a test procedure that, when executed, will provide a set of instructions sufficient to automatically ascertain the operational condition of a test subject and detect and isolate faults within a test subject.

3.1.31 test resource information model (TRIM): Identifies entities and the relationships between the entities of a test equipment configuration. See IEEE Std 1226-1993.

3.1.32 test subject: The physical object or type of physical object that is the focus of attention or target for the development of tests and diagnostics.

3.1.33 unit under test (UUT): Any system, subsystem, assembly, or subassembly undergoing test.

3.1.34 virtual resource: A specific type of resource that defines a representative interface and test behavior between a test equipment configuration and a test subject. More specifically, a virtual resource defines a connection or connection set between the external ports of a test equipment configuration and a test subject, as well as a behavior required to test a specific function of the test subject. A virtual resource makes no assumptions about any real resources or conditions necessary to realize the behavior.

3.1.35 virtual resource group: A virtual resource group defines a collection of virtual resources that are required simultaneously in order to test a specific function of a test subject.

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

3.2 Acronyms

ABBET	A Broad-Based Environment for Test
ATE	automatic test equipment
ATLAS	Abbreviated Test Language for All Systems
ATS	automatic test system
CORBA	Common Object Request Broker Architecture
IDL	Interface Definition Language
OMG	Object Management Group
OO	object oriented
OOA	object-oriented analysis
RM	resource management
TAF	test application framework
TEC	test equipment configuration
TFF	test foundation framework
TPS	test program set
TRIM	test resource information model
UUT	unit under test

4. Description

This clause provides a description of the relationships of this standard to IEEE P1226/D15.3 and its associated standards.

4.1 Objectives

Resource management (RM) provides access to descriptive information about test resource behaviors, test resource interconnections, and the connection of the test subject to the ATE. RM also provides all services needed to perform virtual to real resource mapping, virtual resource management, test equipment configuration management, and real resource management.

4.2 Overview

This standard specifies the interfaces that will enable an ATE independent test program to be implemented on an ATE system. ATE independent implementations may utilize the services specified in this standard to provide a transportable means to bridge the gap between the test procedure and the ATE. Examples of the use of the services defined in this standard can be found in Annex A.

A test program contains requirements for the virtual resources that are needed to test a specific test subject. The services described in this standard map the virtual resources to the real resources on a particular ATE. The test resource information model (TRIM) for this standard is referenced in IEEE P1226/D15.3.

The process of defining the services in this standard involved an object-oriented analysis (OOA) in conjunction with the TRIM. The OOA defines a set of class and object diagrams that identifies classes, objects, and their relationships. These diagrams and an explanation of each are found in Annex B of this standard. From the OOA, the services contained in this standard were defined and appear in Clause 6.

A language-independent implementation of the services is shown in Annex C, written in the Interface Description Language (IDL) defined in OMG CORBA/IIOP Revision 2.1. These service representations are computer processible and identify classes, objects, methods, and method parameters.

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

Clause 6 is divided into three sections as follows:

a) Virtual resource management services
b) Test equipment configuration management services
c) Real resource management services

The virtual resource management services describe services that are available at the virtual test level. These services are divided into services for individual virtual resources and services for groups of virtual resources. It is entirely possible to write a test program using only the services specified in the virtual resource management services subclause (6.2) of this standard. For example, a test programmer may know that an ac signal with a –10 dBm power and a 100 MHz frequency is needed at test subject pin J1-1. This information is conveyed to the virtual services in this standard via the test procedure, which may or may not be written in ATLAS. The examples in Annex A of this standard utilize ATLAS, but this is arbitrary. The test-specific information may be used to create and manipulate a virtual resource via the services defined in the virtual resource management subclause (6.2) of this standard. The virtual resource can be mapped to real resources and, eventually, used to command the real resources to perform the functions needed by the test programmer. An example of this can be found in more detail in Clause A.1 of Annex A. It is important to note that the virtual resource creation and initialization can occur dynamically, during test execution, or statically, at some time prior to test execution.

The virtual resource group services are a subset of the services that are provided in the virtual resource management services subclause (6.2) of this standard. The discriminating factor for the services associated with the virtual resource group is simultaneity. The virtual resource group services let the test programmer group virtual resources together and indicate that all of them must be available at the same time in order for testing to occur. An example that uses the virtual resource group services can be found in Clause A.3 of Annex A.

The test equipment configuration management services describe services to manipulate and search information related to the configuration of a particular set of test equipment. These services can be utilized by the implementation of the virtual resource management services, or they can be accessed directly by tester configuration tools or by the test program itself. An example of a tool that may want to access the test equipment configuration services is a test subject/tester analysis tool. This tool could analyze a particular tester configuration, and could identify whether it could be used to test a specific test subject. The analysis would be based on the behaviors of the test subject that need to be tested. Clause A.9 in Annex A describes just such a tool.

In addition, a test program could directly use the test equipment configuration management services to find a set of real resources and paths that satisfy a particular test subject behavior to be tested. Once a list of real resources and paths are found, a decision can be made on which to use, and these real assets manipulated through the real resource management services of this standard. Clause A.5 in Annex A further describes this scenario.

The real resource management services describe services to find and allocate physical resources and paths. These services are usable from implementations of the virtual resource services or directly from the test program. Clause A.6 in Annex A describes, in detail, how a test program can utilize the real resource management services. In particular, if the test program knows the tester's real resources and paths, the services in the real resource management subclause (6.4) of this standard can be used to allocate, deallocate, and manipulate information about them.

4.3 Relationship to ABBET architecture

The primary purpose of A Broad-Based Environment for Test (ABBET) is to provide for a broad-based environment for test that encompasses the total product life cycle. To do so, an integrated set of interfaces to

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

test related information and services are defined that support the information and implementation needs of this test environment for automated testing applications.

The ABBET architecture is defined by IEEE P1226/D15.3. It defines the existence of interfaces between software components. Each of these interfaces is, in turn, specified in separate ABBET component standards, such as the software interface defined by this standard. The relationship of this standard to IEEE P1226/D15.3 is defined in terms of the ABBET architecture that is defined within IEEE P1226/D15.3.

4.3.1 Architecture levels

The ABBET architecture consists of a set of interfaces that are created from a single class hierarchy that is created in a foundation infrastructure. The foundation provides the basic frame of reference, both to basic classes and to primary interfaces. These can be specialized at different levels to create general test environments (frameworks) or specific test applications. Whereas a library is normally a collection of subroutines, a framework is a collection of classes. Both are usually designed to provide required functionality in a specific and limited application domain.

Figure 1 diagrams the ABBET architecture. The discussion that follows describes each of the following major elements:

a) Foundation framework
b) Extension framework
c) Application

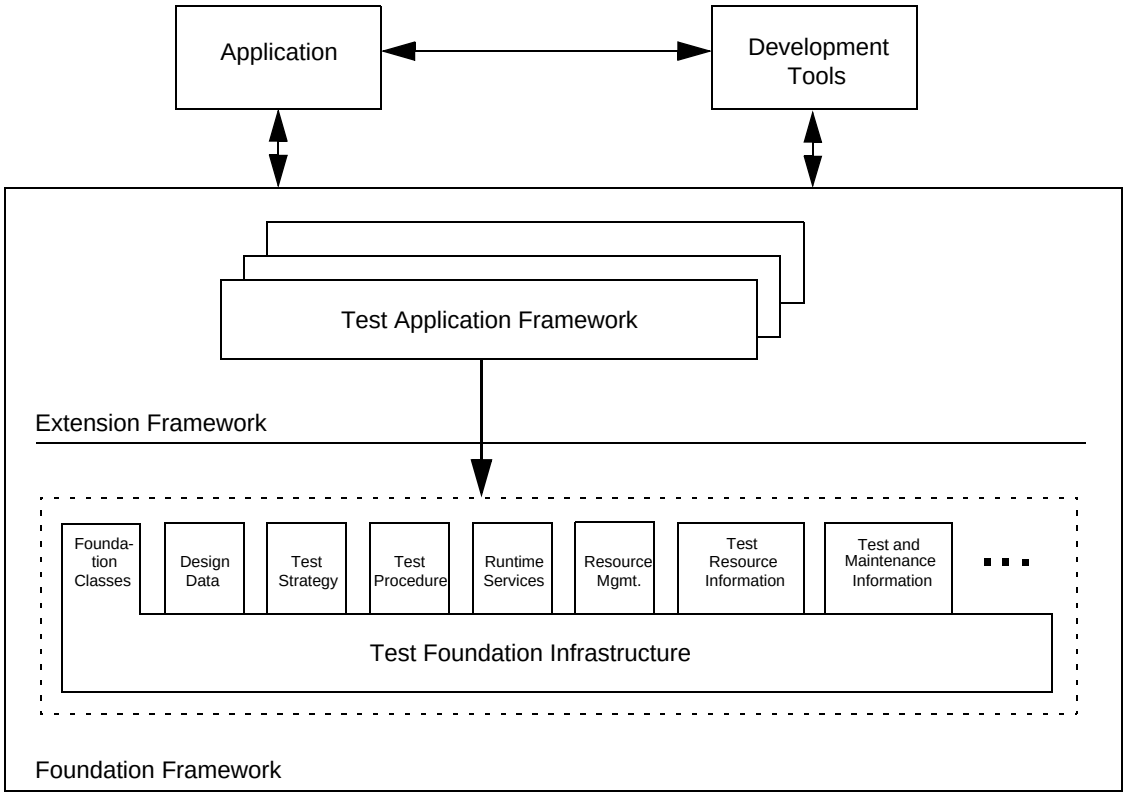

Figure 1—ABBET architecture

--`,,`-`-`,,`,,`,`,,`---
Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

4.3.2 Foundation framework

Any single organization can produce applications for its own needs and according to its own policies, technical expertise, and available test facilities. These applications are useful within that organization, but they usually cannot be shared with others until a basis for that sharing has been achieved. ABBET provides the standards upon which a foundation for object sharing can be built.

The foundation framework is organized as a set of interfaces, with each interface related to one or more ABBET component standards or to external standards maintained by either the IEEE or another recognized body. Basic interfaces appropriate to various aspects of a product life cycle are defined for each product family. The foundation framework contains the basic classes applicable to the components regardless of the application. The other interfaces in the foundation framework use and inherit from the foundation classes.

4.3.3 Extension framework

Foundation classes are too generic and abstract to be of direct use in test application development. However, they provide the starting point for additional specialization that leads to useful test applications. As new classes are developed, new interfaces are defined that select and group objects in support of a product family, mission, or application type. The resulting group is called a test application framework (TAF). A TAF is a reusable collection of classes that is designed to meet the needs of a specific application area. Each TAF provides services to support a specific category of test subjects, technologies, resources, or requirements.

Each class within a TAF fits into the class hierarchy created in the foundation infrastructure. Once the component classes of an application framework have been declared within a test procedure or program, they can be used as if they were a fundamental part of the system.

The extension framework level contains one or more TAFs. TAFs can be moved into the foundation framework if the application is of wide interest and greater accessibility is desired. For example, RM was originally considered a test application framework. It was moved to the foundation framework because of its widespread uses and the need for greater accessibility by other interfaces in this framework.

4.3.4 Application

The ABBET architecture supports the application level by providing direct access both from test development tools and TAFs.

An application may use one or more foundation frameworks to provide access to the classes needed to implement the application. The ABBET object model avoids any naming conflicts between various foundation framework classes and subclasses through the use of OMG IDL module definitions (see OMG CORBA/IIOP Revision 2.1). Each module within the foundation framework has a unique module name. Since an OMG IDL module provides for fully qualified names to be used, each class and method of a class within the module has a unique name. Therefore, only the module names are required to be managed. This mechanism should be applied to any extension framework also.

4.3.5 Relation to ABBET standards

The TRIM provides the information structure needed for a repository of test system information. It can be realized as a database of test instrument capabilities and test system configuration data. Generally, the data in such a model is changed only when an instrument is updated or swapped from an existing configuration. The TRIM permits the information and its access methods to be shared among test systems, vendors, and users, thereby minimizing redundant work.

The RM component provides the primary interfaces needed to fully manage resources found within a test system. It is also a part of the foundation framework. An implementation of this component may use other

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

interfaces or classes defined in the foundation framework such as the TRIM or the interface to instrument drivers or instrument communication buses.

An implementation of this standard is intended to work with or without an implementation of the TRIM. This standard defines services for the creation and utilization of physical objects identified in the TRIM, such as paths, composite resources, and individual resources. Each of the objects in this standard has an initialization service with a name parameter. The purpose of the initialization services in this standard is to link the IEEE Std 1226.3-1998 level objects with their lower-level equivalent objects. If a TRIM implementation is not included, then these same objects will be defined in an implementation-dependent manner. In either case, the IEEE Std 1226.3-1998 level objects become linked to the lower-level objects in order to perform the services in this standard.

4.4 Relationship to reference documents

OMG CORBA/IIOP Revision 2.1 contains the IDL definition that is used to develop the IDL binding for the services in this standard. IEEE Std 1226-1993 is used as a reference for some of the ATLAS-based scenarios and comments used throughout this standard.

5. Conformance

This clause defines the requirements for conformance with this standard. It defines the following:

a) The minimum functionality required
b) Allowed subsets and extensions
c) The approach for implementing the subsets and extensions

An implementation is in conformance with this standard if it provides the minimum required RM functionality described in 5.1. It may also be tailored to one of the allowed subsets and/or extended as specified in 5.2 and 5.3.

Any claim of conformance shall specifically list those functions that are not supported and those extensions that have been made.

5.1 Minimum resource management interface functionality

An implementation shall be in conformance with this standard if it supports one or more of the allowable subsets, as defined in 5.2.

5.2 Subsets

Table 1 identifies the allowed subsets and groups of allowable subsets, along with the required services for each. All services within a subset must be implemented in order to be compliant.

A subset is defined as an implementation in which not every service is included for commercial or technical reasons.

5.3 Extensions

A test system implementation shall be in conformance with this standard if it supports extensions as defined in the following paragraph.

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Table 1—Resource management allowed subsets

Subset	Service groups	Paragraph number
1	Virtual resource	6.2.1– 6.2.4, 6.3.1.11*
2	Virtual resource Virtual resource group	6.2, 6.3.1.10*, 6.3.1.11*
3	Test equipment configuration Real resource management	6.2.2 - 6.2.4, 6.3.1 (except 6.3.1.10 and 6.3.1.11), 6.4
4	Real resource management	6.2.4, 6.4
5	Virtual resource Test equipment configuration Real resource management	6.2.1–6.2.4, 6.3.1 (except 6.3.1.10), 6.4
6	Virtual resource Real resource management	6.2.1– 6.2.4, 6.4
7	Virtual resource Virtual resource group Real resource management	6.2, 6.3.1.10*, 6.3.1.11*, 6.4

*This service is not required for an object-oriented (OO) implementation. If a non-object-oriented implementation is performed, this service is required as well as the create, destroy, and initialize services of test equipment configuration (TEC).

Whenever an existing RM behavior, as defined by the most current, approved version of this standard, can satisfy an application requirement, then existing RM services shall be used. If services are created that provide a legitimate new RM behavior required for the implementation of resource management applications, then any such services shall be identified as extensions.

5.4 Approach for implementation of subsets

All subsets of this standard shall be in accordance with 5.2.

No adaptation or distortion of the service interface(s) or description(s) of this standard is allowed.

A subset can only be created by selecting the required services, as defined in Table 1 of the most current, approved version of this standard. The subset shall be complete (i.e., each of the services indicated in the subset must be implemented).

5.5 Approach for implementation of extensions

Should behavior be required that is not supported by the services defined in this standard, an ABBET Submittal Form (available from the ABBET subcommittee) shall be completed and submitted to the SCC20 ABBET subcommittee for consideration as new material to be incorporated into this standard. All extensions used in an implementation shall be clearly and explicitly identified as changes to this standard.

Extensions are allowed to utilize the standard services; however, the standard services are not allowed to be dependent on the extensions. Users shall be made aware, through clear and explicit documentation, that applications utilizing the extensions will not be portable to other compliant implementations of RM.

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6. Specification

6.1 Conventions

This clause defines the conventions used to describe the services for this standard. Services are grouped by class. For each class, there is a state diagram that describes the methods for state transition. For each service of a class, there is a definition that includes parameters, a list of the types of parameters, a description of the service action, and a description of the completion status returned. Classes are grouped into three segments:

a) Virtual resource management
b) Test equipment configuration management
c) Real resource management

6.1.1 Specification layout

The following shows the structure for the specification of the services.

Format:	Example:
6.x First segment	**6.2 Virtual resource management services**
6.x.1 First class of this segment	**6.2.1 [Virtual Resource]**
6.x.1.1 State diagram for this class	**6.2.1.1 State model**
6.x.1.2 First service of this class	**6.2.1.2 [Virtual Resource] create**
6.x.1.2.1 Interface for this service	**6.2.1.2.1 Service interface**
6.x.1.2.2 Description for this service	**6.2.1.2.2 Description**
6.x.1.2.3 Completion status for this service	**6.2.1.2.3 Completion status**
6.x.1.3 Next service of this class	**6.2.1.3 [Virtual Resource] initialize**
:	**:**
:	**:**
6.x.2 Next class of this segment	**6.2.2 [Connection]**
:	
:	
6.y Next segment	**6.3 Test equipment configuration management services**

6.1.2 State diagram

A state diagram is provided for each class. These diagrams describe the states for each object in a class along with the state transitions that occur as a result of service performance. This serves as a top-level validation that sufficient services have been defined within each class. Each object has an initial state of nonexistent, which means that no object of that class has been created. The create service causes a state transition to the exist state. The exist state occurs when an identifier is established for an object of a given class. The initialize service causes a state transition to some class-dependent state that has a named identifier, and possibly other attributes. Separate states for existence and initialized are required to conform to the parameterless constructors and destructors defined in IDL.

6.1.3 Service definition

Each service is defined by providing the interface definition with parameters and their types, a natural language description of the service, and a completion status. The formal IDL service definitions are provided in Annex C.

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.3.1 Interface and description

This paragraph identifies the format used to describe the language-independent specifications throughout Clause 6. The initial entry of a service name is preceded by the class name enclosed in square brackets. Within the service interface, the object identifier upon which the service is acting is enclosed in triangular brackets preceding the service name. If the service operates on a class instead of an object (such as create), no preceding identifier is applicable. The service interface within each service describes the input and output parameters. Each parameter is listed as an input or an output followed by the specified identifier and its type.

Format:

Service:	service name	
Input:	input parameter name;	input parameter type
Output:	output parameter name;	output parameter type

Example:

Service:	<virtual_resource_id>.perform_service	
Input:	service_id	service_to_perform
Input:	parameters	input_parameters
Output:	measured value[5]	data

6.1.3.2 Type definitions

The parameters are one of the types listed below. Those types that are defined as “a unique object identifier” are the result of creating an object of their associated class. This identifier is used to call all the services of the class, applied to a unique instantiation. For the enumeration, the actual values are identified in the services where they are utilized.

behavior_id	A unique behavior object identifier
connection_id	A unique object identifier
connection_set_id	A unique object identifier
data	A unique object identifier
criteria	Enumerated type to indicate search criteria
input_parameters	A unique object identifier
object_id	A unique object identifier
path_id	A unique object identifier
path_set_id	A unique object identifier
port_id	A unique object identifier
real_resource_id	A unique object identifier
real_resource_set_id	A unique object identifier
service_to_perform	Enumerated type to indicate an action to perform
set_of_behavior_id	A unique object identifier
set_of_connection_ids	A unique object identifier
set_of_path_id	A unique object identifier
set_of_path_sets_id	A unique object identifier
set_of_real_resources_id	A unique object identifier

[5]Optional parameter.

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

status	Enumerated type to indicate availability
string	A sequence of character data, with a minimum length of 0 and no defined maximum length
test_equipment_configuration_id	A unique object identifier
virtual_resource_group_id	A unique object identifier
virtual_resource_id	A unique object identifier

6.1.3.3 Completion status definitions

The completion status subclause of each service identifies all possible error conditions that may be raised by the service. It is assumed a successful completion will exhibit no error or will simply return an expected output value. During the error condition, the class name and service name will be included with the raised error to identify the error's occurrence. When an error occurs, the system shall be left in the calling state. No partial completion of a service is permitted.

For different language implementations, the completion status is realized according to the IDL to language mapping. It may be implemented as an exception, as in Ada, or as a returned or queried error code, as in C. No implementation restriction is made in this clause of the standard.

The completion statuses are listed in alphabetical order, with a brief description of their meaning:

Invalid behavior:	indicates that the behavior_id provided is invalid or not created.
Invalid connection:	indicates that an invalid connection_id has been specified.
Invalid connection set:	indicates that the connection set is invalid or not created.
Invalid name:	indicates an illegal name has been used.
Invalid parameters:	indicates that the object has received invalid parameter values.
Invalid path:	indicates that the path provided is invalid or not created.
Invalid path set:	indicates that the path_set identifier provided is invalid or not created.
Invalid port:	indicates that an invalid port_id has been specified.
Invalid real resource:	indicates that the real resource identifier does not exist.
Invalid real resource set:	indicates that the real resource set provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid service id:	indicates that the service_id parameter is invalid.
Invalid set of path set:	indicates that a set_of_path_set_id provided is invalid or not created.
Invalid set of path sets:	indicates that the set of path sets is invalid.
Invalid test configuration:	indicates that the test equipment configuration provided is invalid or not created.
Invalid virtual resource:	indicates that the virtual resource provided is invalid or not created.
Invalid virtual resource group:	indicates that the virtual resource group provided is invalid or not created.
No behavior:	indicates that no behavior is associated with this path.
Unsuccessful operation:	indicates that the service has failed.

6.2 Virtual resource management services

This subclause defines the functionality of services for the management of virtual test resources.

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.1 [Virtual Resource]

6.2.1.1 State diagram

Figure 2 represents the state diagram for the virtual resource class.

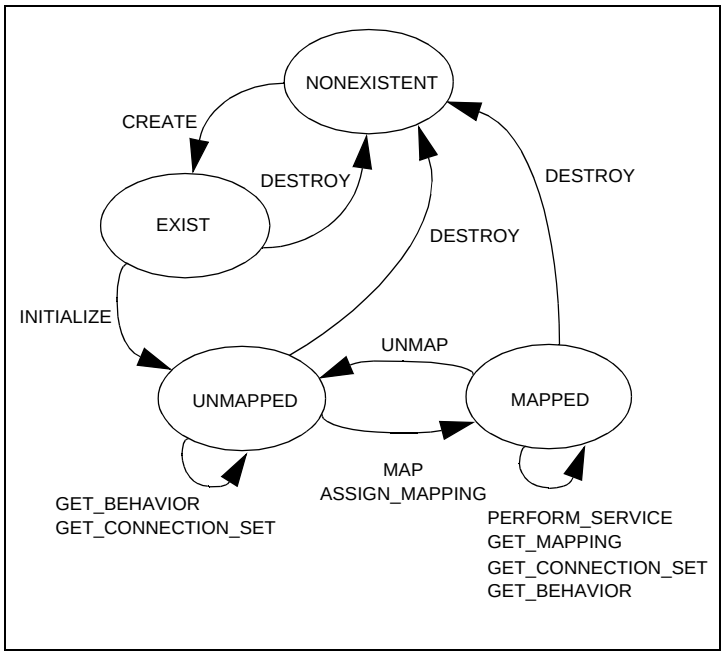

Figure 2—Virtual resource state diagram

6.2.1.2 [Virtual Resource] create

6.2.1.2.1 Service interface

Service: create

Output: virtual resource; virtual_resource_id

6.2.1.2.2 Description

This service shall create a virtual resource object and shall return a unique virtual resource object identifier. No object identifiers shall be duplicated and no duplicate virtual resource names shall be allowed. If the virtual resource cannot be created, a value indicating this shall be returned.

6.2.1.2.3 Completion status

Unsuccessful operation: indicates that the service has failed.

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.1.3 [Virtual Resource]

6.2.1.3.1 Service interface

Service:	<virtual_resource_id>.initialize	
Input:	name;	string
Input:	behavior;	behavior_id
Input:	connection set;	connection_set_id

6.2.1.3.2 Description

This service shall initialize a virtual resource object. A search for candidate resources and paths may be done optionally as a function of this service. No actual mapping or enabling of resources and paths, however, shall be done as a result of this service. The behavior input parameter is a behavior object identifier that represents a particular behavior, such as an ac signal at –5 dBm and 100 MHz. The connection set input parameter is the connection set that is associated with this virtual resource. A connection set is a pair of virtual port identifiers that create a virtual connection.

6.2.1.3.3 Completion status

Invalid name:	indicates that an illegal name has been used.
Invalid behavior:	indicates that the behavior_id provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid connection set:	indicates that the connection set is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.1.4 [Virtual Resource] destroy

6.2.1.4.1 Service interface

Service:	<virtual_resource_id>.destroy

6.2.1.4.2 Description

This service shall destroy a virtual resource by deleting the virtual resource object identified by its unique virtual resource identifier. No disabling of physical resources associated with the virtual resource shall occur.

6.2.1.4.3 Completion status

Invalid virtual resource:	indicates that the virtual resource provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.1.5 [Virtual Resource] map

6.2.1.5.1 Service interface

Service:	<virtual_resource_id>.map

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

6.2.1.5.2 Description

This service shall map a virtual resource by mapping and enabling its behaviors to specific physical paths. This service shall perform this service in one of two ways. First, if candidate real resources and paths have been found previously by the initialize service, this service shall choose among the candidates and perform the mapping. Second, if the candidate resources and paths have not been found, this service shall perform this function first, choose among the candidates, and then perform the mapping.

6.2.1.5.3 Completion status

Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid virtual resource:	indicates that the virtual resource provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.1.6 [Virtual Resource] unmap

6.2.1.6.1 Service interface

Service: <virtual_resource_id>.unmap

6.2.1.6.2 Description

This service shall unmap a virtual resource by releasing the resources and paths. The virtual instrument, however, is still available to be relinked with other resources and paths.

6.2.1.6.3 Completion status

Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid virtual resource:	indicates that the virtual resource provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.1.7 [Virtual Resource] assign mapping

6.2.1.7.1 Service interface

Service: <virtual_resource_id>.assign_mapping

Input:	resource mapping;	real_resource_id
Input:	connection mapping;	set_of_path_sets_id

6.2.1.7.2 Description

This service shall map specific real resources and real paths to a virtual resource. This service differs from the <virtual_resource_id> map service in that the virtual resource does not get to choose which real resources and path to use. The choosing function is typically performed by the virtual resource group, which has access to information for all the virtual resources within that group. The specific mapping information is then passed to this routine via input parameters.

The resource mapping is a real resource object identifier that represents a physical instrument or a physical composite instrument that is to be mapped to the specified virtual resource. The input parameter,

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

connection_mapping, is an object identifier that indicates a set of path sets that represent physical path sets that are to be mapped to the specified virtual resource.

6.2.1.7.3 Completion status

Invalid set of path sets:	indicates that the set of path sets is invalid.
Invalid real resource:	indicates that the real resource identifier does not exist.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid virtual resource:	indicates that the virtual resource provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.1.8 [Virtual Resource] get mapping

6.2.1.8.1 Service interface

Service:	<virtual_resource_id>.get_mapping	
Output:	resource mapping;	real_resource_id
Output:	connection mapping;	set_of_path_sets_id

6.2.1.8.2 Description

This service shall return the physical mapping of the virtual resource to a real resource and paths. The resource mapping output parameter is a real resource object identifier that has been mapped to the specified virtual resource. The connection mapping output parameter is a set of path sets that have been mapped to a specified virtual resource. A path set is a set of path object identifiers. If either output cannot be generated, a value that indicates this will be returned.

6.2.1.8.3 Completion status

Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid virtual resource:	indicates that the virtual resource provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.1.9 [Virtual Resource] perform service

6.2.1.9.1 Service interface

Service:	<virtual_resource_id>.perform_service	
Input:	service id	service_to_perform
Input:	parameters	input_parameters
Output:	measured value[6]	data

[6]Optional parameter.

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

6.2.1.9.2 Description

This service shall cause the behavior associated with a particular virtual resource to occur. The service id input parameter identifies which physical activity is to occur. For example, the virtual resource may be set up as a source, but the service id will indicate if a setup or apply is to be done. The parameter input values are the specific values reserved for the virtual resource, and the optional measured value parameter is a value returned from the real resource. This service can only be called if the virtual resource has previously been mapped, which guarantees the associated behavior can be performed with specific real resources.

6.2.1.9.3 Completion status

Invalid virtual resource:	indicates that the virtual resource provided is invalid or not created.
Invalid service id:	indicates that the virtual resource cannot provide this function.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid parameters:	indicates that the virtual resource has received invalid parameter values.
Unsuccessful operation:	indicates that the service has failed.

6.2.1.10 [Virtual Resource] get behavior

6.2.1.10.1 Service interface

Service: <virtual_resource_id>.get_behavior

Output: behavior; behavior_id

6.2.1.10.2 Description

This service returns the behavior id with which the virtual resource object was initialized.

6.2.1.10.3 Completion status

Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.2.1.11 [Virtual Resource] get connection set

6.2.1.11.1 Service interface

Service: <virtual_resource_id>.get_connection_set

Output: connection set; connection_set_id

6.2.1.11.2 Description

This service returns the connection set id with which the virtual resource object was initialized.

6.2.1.11.3 Completion status

Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

6.2.2 [Connection]

6.2.2.1 State diagram

Figure 3 represents the state diagram for the connection class.

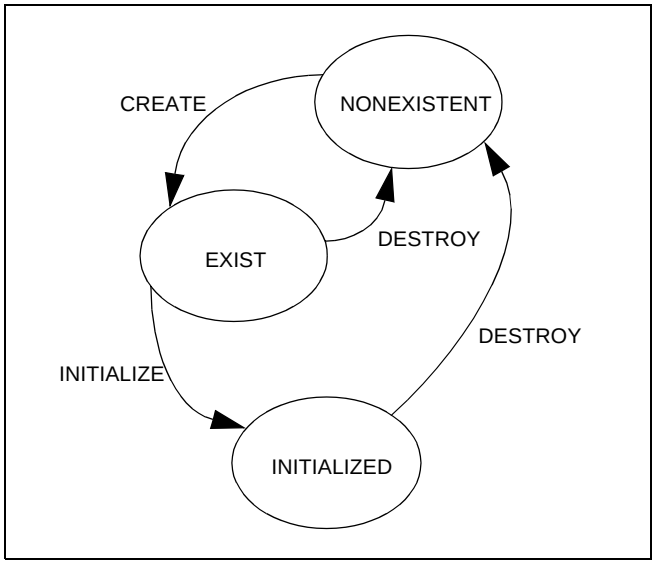

Figure 3—Connection state diagram

6.2.2.2 [Connection] create

6.2.2.2.1 Service interface

Service:	create	
Output:	connection;	connection_id

6.2.2.2.2 Description

This service shall create a connection. If the connection cannot be created, a value indicating this shall be returned. A single connection identifier is output, which is a reference identifier to access the connection in the future.

6.2.2.2.3 Completion status

Unsuccessful operation: indicates that the service has failed.

6.2.2.3 [Connection] initialize

6.2.2.3.1 Service interface

Service:	<connection_id>.initialize	
Input:	name;	string
Input:	from port;	port_id
Input:	to port;	port_id

--`,,-`-`,,`,,`,`,,`---

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

6.2.2.3.2 Description

This service shall initialize a previously created connection with the two ports specified.

6.2.2.3.3 Completion status

Invalid name:	indicates that an illegal name has been specified.
Invalid connection:	indicates that an invalid connection_id has been specified.
Invalid port:	indicates that an invalid port_id has been specified.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.2.2.4 [Connection] destroy

6.2.2.4.1 Service interface

Service: <connection_id>.destroy

6.2.2.4.2 Description

This service shall destroy the connection by deleting its unique identifier.

6.2.2.4.3 Completion status

Invalid connection:	indicates that the connection provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.3 [Connection Set]

6.2.3.1 State diagram

Figure 4 represents the state diagram for the connection set class.

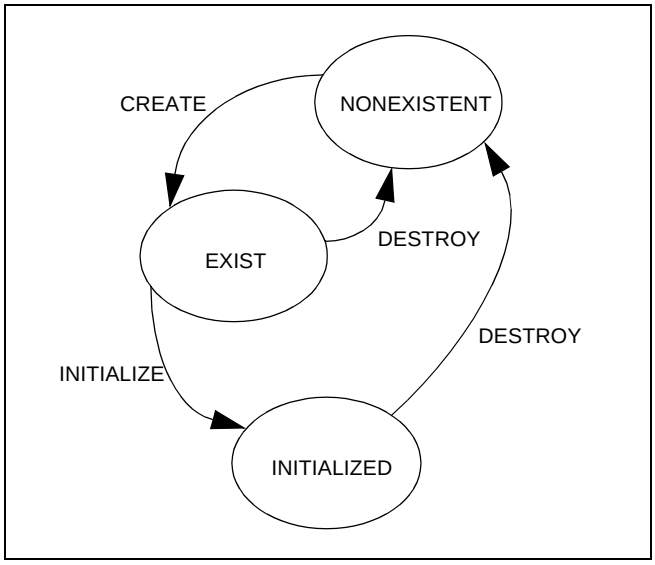

Figure 4—Connection set state diagram

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.3.2 [Connection Set] create

6.2.3.2.1 Service interface

Service: create

Output: connection set; connection_set_id

6.2.3.2.2 Description

This service creates an object of the class called connection set. If the connection set cannot be created, a value indicating this shall be returned. The connection set inherits from the connection set that is defined in the test foundation framework (TFF). Services that are inherited include add and remove; however, a locking mechanism is included to restrict users from inadvertently changing a connection set during utilization. Returned from this service is a connection set identifier that is used to access the set.

6.2.3.2.3 Completion status

Unsuccessful operation: indicates that the service has failed.

6.2.3.3 [Connection Set] destroy

6.2.3.3.1 Service interface

Service: <connection_set_id>.destroy

6.2.3.3.2 Description

This service shall destroy the connection set by deleting its unique identifier.

6.2.3.3.3 Completion status

Invalid connection set: indicates that the connection provided is invalid or not created.
Unsuccessful operation: indicates that the service has failed.

6.2.4 [Port]

6.2.4.1 State diagram

Figure 5 represents the state diagram for the port class.

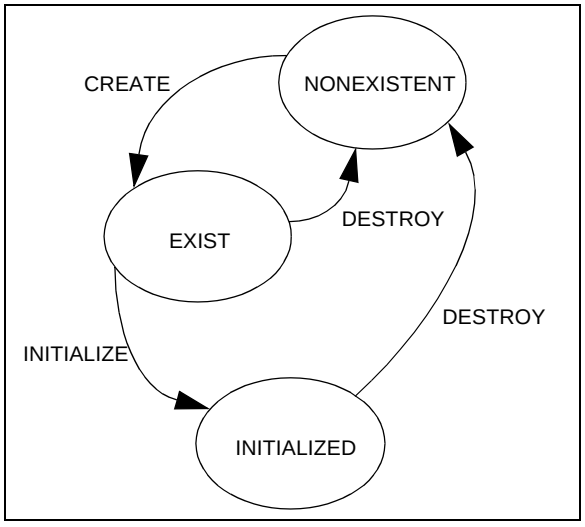

Figure 5—Port state diagram

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.4.2 [Port] create

6.2.4.2.1 Service interface

Service: create

Output: port; port_id

6.2.4.2.2 Description

This service shall create a port object and shall return a unique port object identifier. The output parameter, port, is an identifier that represents the actual port object and is unique to a port object. If the port cannot be created, a value that indicates this shall be returned. The object identifier is the only way to access a port object.

6.2.4.2.3 Completion status

Unsuccessful operation: indicates that the service has failed.

6.2.4.3 [Port] initialize

6.2.4.3.1 Service interface

Service: <port_id>.initialize

Input: name; string

6.2.4.3.2 Description

This service shall initialize a previously created port object. The input parameter, name, is a string that represents the physical name of the port.

6.2.4.3.3 Completion status

Invalid name: indicates that an illegal name has been used.
Invalid port: indicates that the port provided is invalid or not created.
Invalid state: indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation: indicates that the service has failed.

6.2.4.4 [Port] destroy

6.2.4.4.1 Service interface

Service: <port_id>.destroy

6.2.4.4.2 Description

This service shall destroy a port object by deleting its unique identifier.

6.2.4.4.3 Completion status

Invalid port: indicates that the port provided is invalid or not created.
Unsuccessful operation: indicates that the service has failed.

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.5 [Virtual Resource Group]

6.2.5.1 State diagram

Figure 6 represents the state diagram for the virtual resource group class.

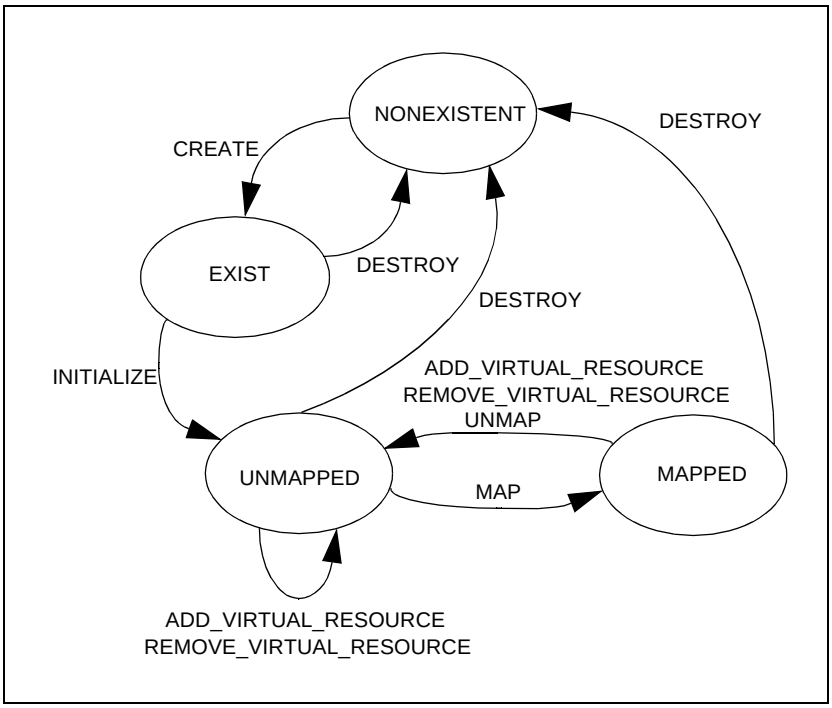

Figure 6—Virtual resource group state diagram

6.2.5.2 [Virtual Resource Group] create

6.2.5.2.1 Service interface

Service: create

Output: virtual resource group; virtual_resource_group_id

6.2.5.2.2 Description

This service shall create a virtual resource group that is a container structure that allows a grouping of virtual resources to occur. This service shall return a unique virtual resource group identifier that represents the created group. The virtual resource group identifier is the only way to access the group and its resources. If the virtual resource group cannot be created, a value indicating this shall be returned.

6.2.5.2.3 Completion status

Unsuccessful operation: indicates that the service has failed.

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.5.3 [Virtual Resource Group] initialize

6.2.5.3.1 Service interface

Service: <virtual_resource_group_id>.initialize

Input name; string

6.2.5.3.2 Description

This service shall initialize a previously created virtual resource group. The input parameter is a string that represents the virtual resource group name. Should the creation of this virtual resource group happen outside the domain of a test application, the test application shall use the name during initialization to retrieve the identifier later via a call to get_virtual_group_id.

6.2.5.3.3 Completion status

Invalid virtual resource group:	indicates that the virtual resource group provided is invalid or not created.
Invalid name:	indicates that an illegal name was used as a parameter.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.2.5.4 [Virtual Resource Group] destroy

6.2.5.4.1 Service interface

Service: <virtual_resource_group_id>.destroy

6.2.5.4.2 Description

This service shall destroy a virtual resource group object by deleting its unique identifier. No disabling of physical resources associated with any of the virtual resources within the group shall occur.

6.2.5.4.3 Completion status

Invalid virtual resource group:	indicates that the virtual resource group provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.5.5 [Virtual Resource Group] map

6.2.5.5.1 Service interface

Service: <virtual_resource_group_id>.map

6.2.5.5.2 Description

This service shall map physical entities to all virtual resources that are associated with the virtual resource group that is identified by the virtual_resource_group_id. This could consist of mapping all virtual resources of the group or only those that were added since the last mapping. It is left to each implementation of the virtual resources to cause the behavior to occur.

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

6.2.5.5.3 Completion status

Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid virtual resource group:	indicates that the virtual resource group provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.5.6 [Virtual Resource Group] unmap

6.2.5.6.1 Service interface

Service: <virtual_resource_group_id>.unmap

6.2.5.6.2 Description

This service shall unmap a group of virtual resources that belong to a virtual resource group. This service shall unmap all virtual resources associated with the virtual resource group that is specified by the virtual resource group identifier.

6.2.5.6.3 Completion status

Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid virtual resource group:	indicates that the virtual resource group provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.2.5.7 [Virtual Resource Group] add virtual resource

6.2.5.7.1 Service interface

Service: <virtual_resource_group_id>.add_virtual_resource

Input: virtual resource; virtual_resource_id

6.2.5.7.2 Description

This service shall add a virtual resource to a virtual resource group. The virtual resource to be added is identified by its unique virtual resource object identifier.

6.2.5.7.3 Completion status

Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid virtual resource group:	indicates that the virtual resource group provided is invalid or not created.
Invalid virtual resource:	indicates that the virtual resource provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

6.2.5.8 [Virtual Resource Group] remove virtual resource

6.2.5.8.1 Service interface

Service: <virtual_resource_group_id>.remove_virtual_resource

Input: virtual resource; virtual_resource_id

6.2.5.8.2 Description

This service shall remove a virtual resource from a virtual resource group. The virtual resource to be removed is identified by its unique virtual resource object identifier. If the virtual resource group is in the mapped state, this service causes a transition to the unmapped state.

6.2.5.8.3 Completion status

Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid virtual resource group:	indicates that the virtual resource group provided is invalid or not created.
Invalid virtual resource:	indicates that the virtual resource provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.3 Test equipment configuration management services

6.3.1 [Test Equipment Configuration]

6.3.1.1 State diagram

Figure 7 represents the state diagram for the test equipment configuration class.

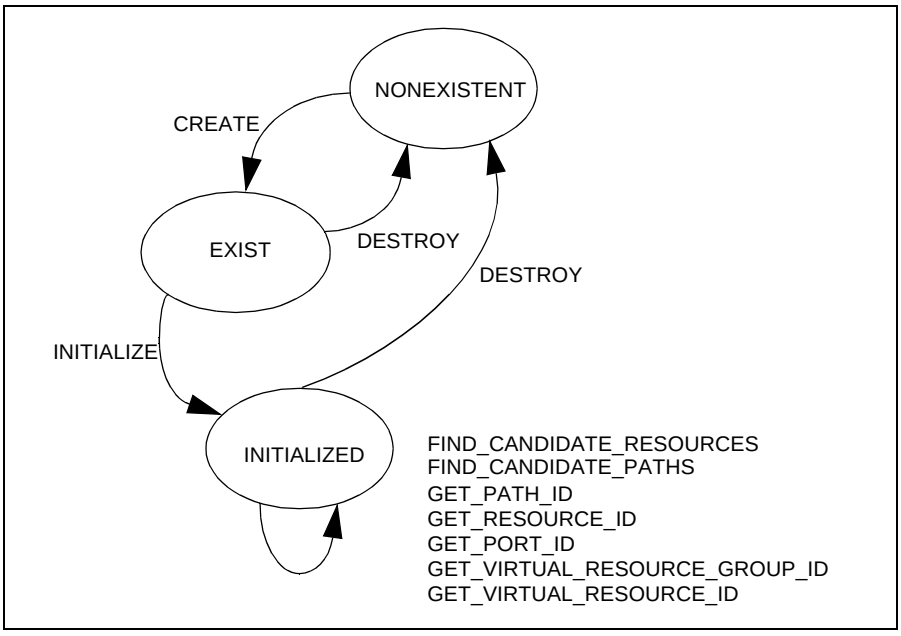

Figure 7—Test equipment configuration state diagram

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.3.1.2 [Test Equipment Configuration] create

6.3.1.2.1 Service interface

Service: create

Output: test equipment configuration; test_equipment_configuration_id

6.3.1.2.2 Description

This service shall create a test equipment configuration object and return a unique test equipment configuration object identifier. The output parameter is an identifier that represents the actual test equipment configuration object and is unique to that test equipment configuration object. If the test equipment configuration cannot be created, a value that indicates this shall be returned. The object identifier is the only way to access the test equipment configuration object.

6.3.1.2.3 Completion status

Unsuccessful operation: indicates that the service has failed.

6.3.1.3 [Test Equipment Configuration] initialize

6.3.1.3.1 Service interface

Service: <test_equipment_configuration_id>.initialize

Input: name; string

6.3.1.3.2 Description

This service shall initialize a previously created test equipment configuration object. The input parameter, name, is a string that represents the physical name of the test equipment configuration.

6.3.1.3.3 Completion status

Invalid name: indicates that an illegal name has been used.
Invalid state: indicates that the service cannot be performed while the object is in its current state.
Invalid test configuration: indicates that the test equipment configuration provided is invalid or not created.
Unsuccessful operation: indicates that the service has failed.

6.3.1.4 [Test Equipment Configuration] destroy

6.3.1.4.1 Service interface

Service: <test_equipment_configuration_id>.destroy

6.3.1.4.2 Description

This service shall destroy a test equipment configuration object by deleting its unique identifier.

--`,,`-`-`,,`,,`,`,,`---
Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

6.3.1.4.3 Completion status

Invalid test configuration:	indicates that the test equipment configuration provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.3.1.5 [Test Equipment Configuration] find candidate resources

6.3.1.5.1 Service interface

Service:	<test_equipment_configuration_id>.find_candidate_resources	
Input:	behavior;	behavior_id
Input:	search criteria;	criteria (all_resources, non_allocated)
Output:	resources;	set_of_real_resources_id

6.3.1.5.2 Description

The find candidate resources service shall search a particular test equipment configuration for all real resources that match the specified behavior. The service shall return a set of real resource identifiers that indicates the matching real resources.

The behavior input parameter is an identifier that specifies a particular behavior object, such as an ac signal at –5 dBm and 100 MHz. The search criteria input parameter provides the conditions for which to search. The two possible conditions are all resources or only nonallocated resources. The output parameter, resources, is a set of all resource identifiers that represent real resources that match the requested behavior and the search criteria.

6.3.1.5.3 Completion status

Invalid behavior:	indicates that the behavior identifier is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid test configuration:	indicates that the test equipment configuration provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.3.1.6 [Test Equipment Configuration] find candidate paths

6.3.1.6.1 Service interface

Service:	<test_equipment_configuration_id>.find_candidate_paths	
Input:	resource;	real_resource_id
Input:	required connection;	connection_set_id
Output:	candidate paths;	set_of_path_sets_id

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.3.1.6.2 Description

The find candidate paths service shall search a particular test equipment configuration for all path sets that match the specified connectivity for a specified real resource. The service shall return a set of path sets that indicates the matching real paths.

The resource input parameter is an identifier that specifies a real resource. The required connection input parameter is a connection set. The output parameter, candidate paths, is a set of path sets that implements the required connection sets for a particular real resource.

6.3.1.6.3 Completion status

Invalid connection set:	indicates that the connection set is invalid.
Invalid real resource:	indicates that no real resource identifier exists for this test equipment configuration and connection set.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid test configuration:	indicates that the test equipment configuration provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.3.1.7 [Test Equipment Configuration] get path id

6.3.1.7.1 Service interface

Service:	<test_equipment_configuration_id>.get_path_id	
Input:	required end points;	connection
Output:	path;	path_id

6.3.1.7.2 Description

This service retrieves the path identifier associated with the required ports that were identified in the input. If no path has been created, a null value is returned. This service interface is identical to the create path service interface, but the functionality is different. This service does not perform a creation function, but only searches for the identifier associated with a particular path. This service is best used when the create path service is utilized outside the visibility of the current application. This service maps the input parameter, required end points, to a specific path and returns the associated path_id in the output parameter. If no path is found, an appropriate identifier is returned.

6.3.1.7.3 Completion status

Invalid connection:	indicates that an invalid connection was specified.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid test configuration:	indicates that the test equipment configuration provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.3.1.8 [Test Equipment Configuration] get resource id

6.3.1.8.1 Service interface

Service:	<test_equipment_configuration_id>.get_resource_id	
Input:	name;	string
Output:	real resource;	real_resource_id

6.3.1.8.2 Description

This service retrieves the resource identifier associated with a particular test equipment configuration. This service is best used when the create resource service is utilized outside the visibility of the current application. This service searches through the set of specific real resources within the test equipment configuration and returns the associated identifier of the matching real resource. If no real resource is found, an appropriate identifier is returned.

6.3.1.8.3 Completion status

Invalid name:	indicates that an illegal name has been used.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid test configuration:	indicates that the test equipment configuration provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.3.1.9 [Test Equipment Configuration] get port id

6.3.1.9.1 Service interface

Service:	<test_equipment_configuration_id>.get_port_id	
Input:	name;	string
Output:	port;	port_id

6.3.1.9.2 Description

This service retrieves the identifier associated with a port within a particular test equipment configuration. This service is best used when the create port service is utilized outside the visibility of the current application. This service searches through the set of specific ports within the test equipment configuration and returns the associated identifier of the matching port. If no port is found, an appropriate identifier is returned.

6.3.1.9.3 Completion status

Invalid name:	indicates that an illegal name has been used.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Invalid test configuration:	indicates that the test equipment configuration provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.3.1.10 [Test Equipment Configuration] get virtual resource group id

6.3.1.10.1 Service interface

Service:	<test_equipment_configuration_id>.get_virtual_resource_group_id	
Input:	name;	string
Output:	virtual resource group;	virtual_resource_group_id

6.3.1.10.2 Description

This service shall retrieve the virtual resource group identifier of a virtual resource group. This service is typically used when the virtual resource group has been created outside the domain of the test application (e.g., at pre-runtime). The output parameter, virtual resource group, is the identifier used to access a virtual resource group.

The input parameter, name, is the name of the virtual resource group that is used during creation. The output parameter, virtual resource group, is the virtual resource group identifier.

6.3.1.10.3 Completion status

Invalid name:	indicates that an illegal name has been used.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.3.1.11 [Test Equipment Configuration] get virtual resource id

6.3.1.11.1 Service interface

Service:	<test_equipment_configuration_id>.get_virtual_resource_id	
Input:	name;	string
Output:	virtual resource;	virtual_resource_id

6.3.1.11.2 Description

This service shall retrieve the virtual resource identifier of a virtual resource. This service is typically used when the virtual resource has been created outside the domain of the test application (e.g., at pre-runtime when the application is running at runtime).

The input parameter, name, is the name of the virtual resource used during creation. The output parameter, virtual resource, is the virtual resource identifier.

6.3.1.11.3 Completion status

Invalid name:	indicates that an illegal name has been used.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

6.4 Real resource management services

This subclause defines the functionality of services for the physical management of test resources.

6.4.1 [Path]

6.4.1.1 State diagram

Figure 8 represents the state diagram for the path class.

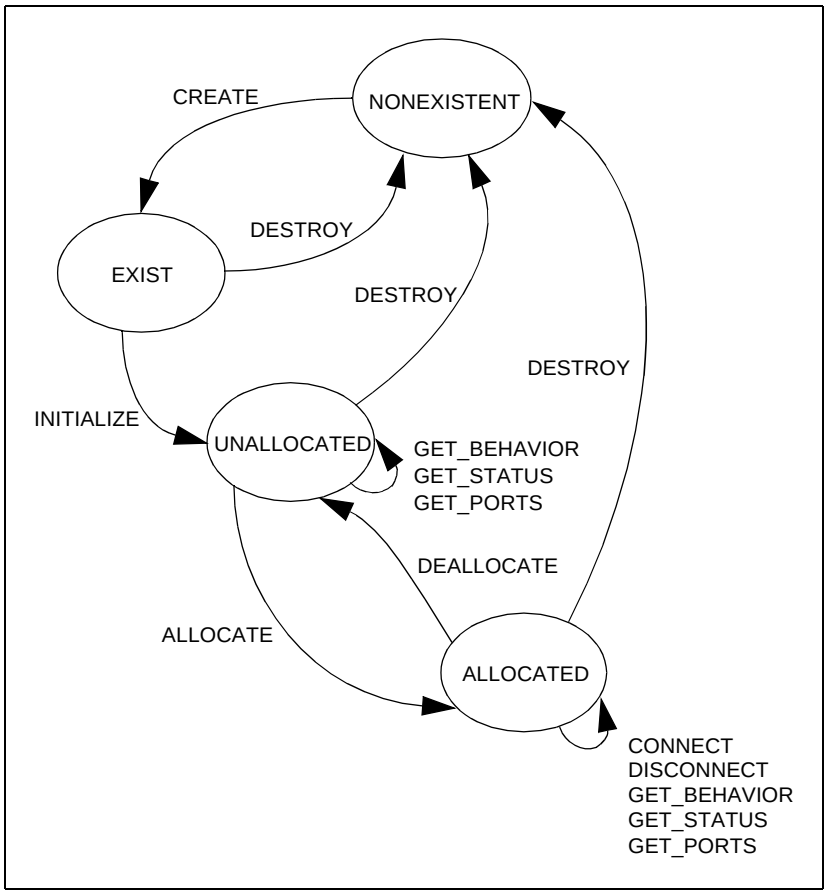

Figure 8—Path state diagram

6.4.1.2 [Path] create

6.4.1.2.1 Service interface

Service:	create	
Output:	path;	path_id

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.4.1.2.2 Description

This service shall create a path object and return a unique path object identifier. The output parameter is a path object identifier and is unique to a particular path object. If the path cannot be created, a value that indicates this shall be returned. The object identifier is the only way to access a path object.

6.4.1.2.3 Completion status

Unsuccessful operation: indicates that the service has failed.

6.4.1.3 [Path] initialize

6.4.1.3.1 Service interface

Service: <path_id>.initialize

Input: name; string

6.4.1.3.2 Description

This service shall initialize a previously created path object. The name input parameter is a string that represents the name of the path. This service will link, via the name parameter, to a real object within the test equipment configuration.

6.4.1.3.3 Completion status

Invalid name: indicates that an illegal name has been used.
Invalid state: indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation: indicates that the service has failed.

6.4.1.4 [Path] destroy

6.4.1.4.1 Service interface

Service: <path_id>.destroy

6.4.1.4.2 Description

This service shall destroy a path object by deleting the object and its reference identifier. If the object is still allocated, all physical resources associated with the object to be destroyed will be deallocated.

6.4.1.4.3 Completion status

Invalid path: indicates that the path provided is invalid or not created.
Unsuccessful operation: indicates that the service has failed.

6.4.1.5 [Path] get behavior

6.4.1.5.1 Service interface

Service: <path_id>.get_behavior

Output: behavior; behavior_id

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

6.4.1.5.2 Description

This service shall obtain the behavior of a specified real path. This service shall be used by the test system when it is searching for candidate real paths that match a required behavior of a virtual resource. The output parameter, behavior, is an identifier that represents a particular behavior object. The behavior object describes a specific behavior with respect to certain characteristics, such as power level or frequency.

6.4.1.5.3 Completion status

Invalid path:	indicates that the path provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
No behavior:	indicates that no behavior is associated with this path.
Unsuccessful operation:	indicates that the service has failed.

6.4.1.6 [Path] allocate

6.4.1.6.1 Service interface

Service: <path_id>.allocate

6.4.1.6.2 Description

This service shall allocate a physical path and make it unavailable for use by any other object. This allocation may come from a virtual resource request or directly from a test program.

6.4.1.6.3 Completion status

Invalid path:	indicates that the path provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.4.1.7 [Path] deallocate

6.4.1.7.1 Service interface

Service: <path_id>.deallocate

6.4.1.7.2 Description

This service shall deallocate a physical path and make it available for use by other objects. This deallocation may come from a virtual resource request or directly from a test program.

6.4.1.7.3 Completion status

Invalid path:	indicates that the path provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.4.1.8 [Path] get status

6.4.1.8.1 Service interface

Service:	<path_id>.get_status	
Output:	path status;	status (available, unavailable)

6.4.1.8.2 Description

This service shall retrieve the status of a particular path object. The output parameter, status, is an enumeration type (available or unavailable) that indicates the status of a path object.

6.4.1.8.3 Completion status

Invalid path:	indicates that the path provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service failed.

6.4.1.9 [Path] get ports

6.4.1.9.1 Service interface

Service:	<path_id>.get_ports	
Output:	end points;	port1_id, port2_id

6.4.1.9.2 Description

This service shall retrieve the ports for a path object. The output parameters, end points, are the port identifiers on either end of the path.

6.4.1.9.3 Completion status

Invalid path:	indicates that the path provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.4.1.10 [Path] connect

6.4.1.10.1 Service interface

Service:	<path_id>.connect

6.4.1.10.2 Description

This service causes an initialized path to be physically created when the path contains switches, relays, or other active components. If this service is invoked on a path that does not contain active components, it shall complete without reporting an error.

--`,,`-`-`,,`,,`,`,,`---
Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

6.4.1.10.3 Completion status

Invalid path:	indicates that the path provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.4.1.11 [Path] disconnect

6.4.1.11.1 Service interface

Service: <path_id>.disconnect

6.4.1.11.2 Description

This service causes all active components in the path to return to their quiescent or normal state. If this service is invoked on a path that does not contain active components, it shall complete without reporting an error.

6.4.1.11.3 Completion status

Invalid path:	indicates that the path provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.4.2 [Path Set]

6.4.2.1 State diagram

Figure 9 represents the state diagram for the path set class.

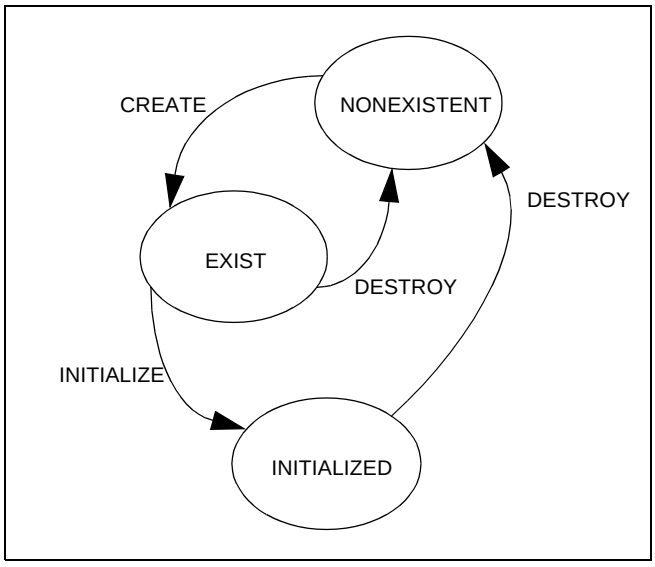

Figure 9—Path set state diagram

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.4.2.2 [Path Set] create

6.4.2.2.1 Service interface

Service:	create	
Output:	path set;	path_set_id

6.4.2.2.2 Description

This service creates an object of the class called path set. If the path set cannot be created, a value that indicates this shall be returned.

Returned from this service is a path set identifier that is used to access the set.

6.4.2.2.3 Completion status

Unsuccessful operation:	indicates that the service has failed.

6.4.2.3 [Path Set] initialize

6.4.2.3.1 Service interface

Service:	<path_set_id>.initialize	
Input:	name;	string
Input:	paths;	set (path)

6.4.2.3.2 Description

This service shall initialize a previously created path set object. The name parameter identifies a physical name associated with this path set. The parameter, paths, of type set (path) is created prior to calling this service. The set (path) inherits from the set class that is defined in the TFF collection module. This set class provides the following functions:

a) Add an item to a set.
b) Remove an item from the set.
c) Check if an item is a member of the set.
d) Retrieve the next item from the set.

A more detailed explanation of the set class is provided in the TFF description. Once initialized with a set (path) parameter, the path set should not be changed except through another call to initialize.

6.4.2.3.3 Completion status

Invalid name:	indicates that an illegal name has been specified.
Invalid path:	indicates that one of the path identifiers provided is invalid or not created.
Invalid path set:	indicates that the path_set identifier provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.4.2.4 [Path Set] destroy

6.4.2.4.1 Service interface

Service: <path_set_id>.destroy

6.4.2.4.2 Description

This service shall destroy a path set object by deleting its unique identifier.

6.4.2.4.3 Completion status

Invalid path set: indicates that the path_set identifier provided is invalid or not created.
Unsuccessful operation: indicates that the service has failed.

6.4.3 [Set of Path Sets]

6.4.3.1 State diagram

Figure 10 represents the state diagram for the set of path sets class.

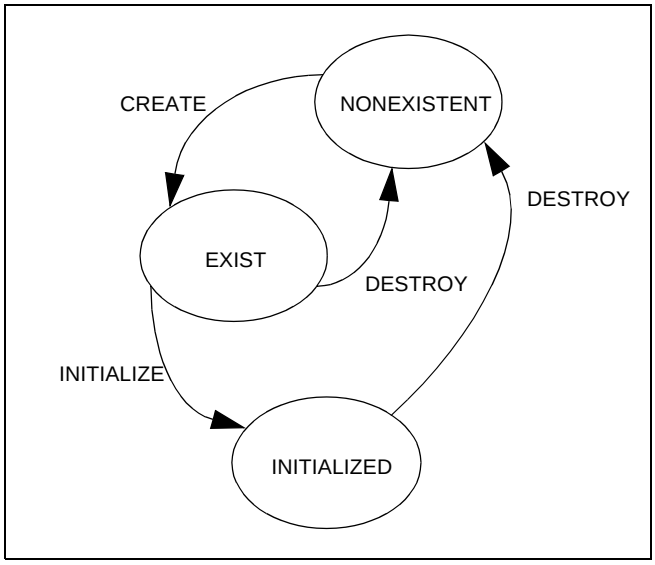

Figure 10—Set of path sets state diagram

6.4.3.2 [Set of Path Sets] create

6.4.3.2.1 Service interface

Service: create

Output: set of path sets; set_of_path_sets_id

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.4.3.2.2 Description

This service creates an object of the set of path sets class. If the set of path sets cannot be created, a value that indicates this shall be returned. The set of path sets, because it is a set, has predefined functions. These functions are as follows:

a) Add a path set to the set of path sets.
b) Remove a path set from the set of path sets.
c) Check if a path set is a member of the set of path sets.
d) Retrieve the next item in the set of path sets.

Returned from this service is a set of path sets identifier that is used to access the set.

6.4.3.2.3 Completion status

Unsuccessful operation: indicates that the service has failed.

6.4.3.3 [Set of Path Sets] initialize

6.4.3.3.1 Service interface

Service:	<set_of_path_sets_id>.initialize	
Input:	name;	string
Input:	path sets	set (path_sets)

6.4.3.3.2 Description

This service shall initialize a previously created set of path sets object. The name parameter identifies a physical name associated with this set of path sets. The parameter, path sets, of type set (path_set) is created prior to calling this service. The set (path_set) inherits from the set class that is defined in the TFF collection module. This set class provides the following functions:

a) Add an item to a set.
b) Remove an item from the set.
c) Check if an item is a member of the set.
d) Retrieve the next item from the set.

A more detailed explanation of the set class can be found in the TFF description. Once initialized with a set (path set) parameter, the set of path sets should not be changed, except through another call to initialize.

6.4.3.3.3 Completion status

Invalid name:	indicates that an illegal name has been specified.
Invalid path set:	indicates that the path_set identifier provided is invalid or not created.
Invalid set of path sets:	indicates that a set_of_path_sets_id provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

--`,,-`-`,,`,,`,`,,`---
Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

6.4.3.4 [Set of Path Sets] destroy

6.4.3.4.1 Service interface

Service: <set_of_path_sets_id>.destroy

6.4.3.4.2 Description

This service shall destroy the set of path sets by deleting its unique identifier.

6.4.3.4.3 Completion status

Invalid set of path sets: indicates that the set_of_path_sets identifier provided is invalid or not created.
Unsuccessful operation: indicates that the service has failed.

6.4.4 [Real Resource]

6.4.4.1 State diagram

Figure 11 represents the state diagram for the real resource class.

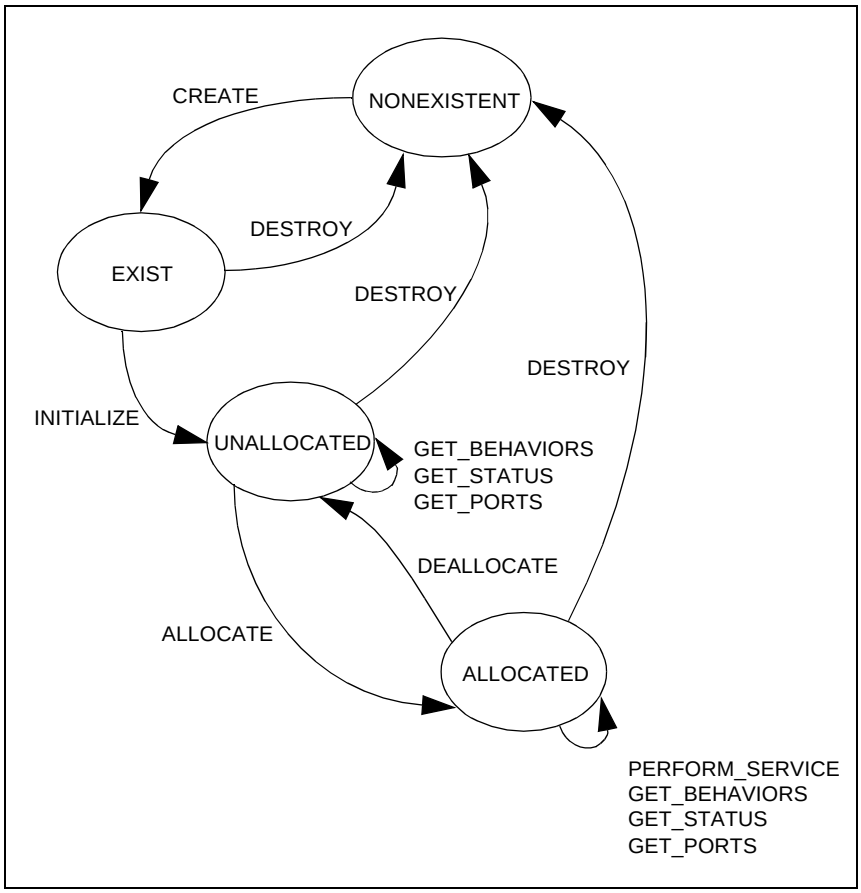

Figure 11—Real resource state diagram

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.4.4.2 [Real Resource] create

6.4.4.2.1 Service interface

Service: create

Output: real resource; real_resource_id

6.4.4.2.2 Description

This service shall create a real resource object and return a unique real resource object identifier. If the real resource cannot be created, a value that indicates this shall be returned. The output parameter, real resource, is an identifier that represents the real resource object. The identifier is the real resource object identifier and is unique to a real resource object. The object identifier is the only way to access the real resource object. As defined in the definition of terms, a real resource is an abstract object that can be either an individual resource or a composite resource.

6.4.4.2.3 Completion status

Unsuccessful operation: indicates that the service has failed.

6.4.4.3 [Real Resource] initialize

6.4.4.3.1 Service interface

Service: <real_resource_id>.initialize

Input: name; string

6.4.4.3.2 Description

This service shall initialize a previously created real resource object. The input parameter, name, is a string representing the physical name of the real resource. This service links the IEEE Std 1226.3-1998 object with the behavior defined in the TRIM or TRIM-like model being used.

6.4.4.3.3 Completion status

Invalid name: indicates that an illegal name has been specified.
Invalid real resource: indicates that the real resource provided is invalid or not created.
Invalid state: indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation: indicates that the service has failed.

6.4.4.4 [Real Resource] destroy

6.4.4.4.1 Service interface

Service: <real_resource_id>.destroy

6.4.4.4.2 Description

This service shall destroy a real resource object by deleting its unique identifier. If the real resource is still allocated, it will be deallocated through the deallocate service before it is destroyed.

--`,,`-`-`,,`,,`,`,,`---
Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

6.4.4.4.3 Completion status

Invalid real resource:	indicates that the real resource provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

6.4.4.5 [Real Resource] get behaviors

6.4.4.5.1 Service interface

Service:	<real_resource_id>.get_behaviors	
Output:	behaviors;	set_of_behavior_id

6.4.4.5.2 Description

This service shall obtain the set of behaviors of a specified real resource. This service shall be used by the test system when it is searching for candidate real resources that match a required behavior of a virtual resource. The output parameter, behaviors, is a set of behavior object identifiers, with each identifier representing a particular behavior object. The behavior object describes a specific behavior, such as an ac signal at –5 dBm and 100 MHz.

6.4.4.5.3 Completion status

Invalid real resource:	indicates that the real resource provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.4.4.6 [Real Resource] allocate

6.4.4.6.1 Service interface

Service:	<real_resource_id>.allocate

6.4.4.6.2 Description

This service shall allocate a real resource. This process shall capture a real resource and make it unavailable for use by any other object. This allocation may come from a virtual resource request or directly from a test program.

6.4.4.6.3 Completion status

Invalid real resource:	indicates that the real resource provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.4.4.7 [Real Resource] deallocate

6.4.4.7.1 Service interface

Service:	<real_resource_id>.deallocate

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.4.4.7.2 Description

This service shall deallocate a real resource and make it available for use by other objects. This deallocation may come from a virtual resource request or directly from a test program.

6.4.4.7.3 Completion status

Invalid real resource:	indicates that the real resource provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.4.4.8 [Real Resource] get status

6.4.4.8.1 Service interface

Service:	<real_resource_id>.get_status	
Output:	real resource status;	status (available, unavailable)

6.4.4.8.2 Description

This service shall retrieve the status of a real resource object. Get status uses the attributes allocation, presence, and health to identify whether a real resource is available or unavailable. If an instrument is uncalibrated, it is assumed that it is unhealthy and unavailable. Get status is a global access function to return the availability of a real resource based on the aforementioned attributes.

6.4.4.8.3 Completion status

Invalid real resource:	indicates that the real resource provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.4.4.9 [Real Resource] get ports

6.4.4.9.1 Service interface

Service:	<real_resource_id>.get_ports	
Input:	port;[7]	port_id
Output:	port set;	set_of_ports

6.4.4.9.2 Description

This service shall accept an optional input port and shall return a set of all ports to which it can be connected. The algorithm considers all ports that can be accessed through the real resource, as indicated by the real_resource_id. If no port is given as an input, this service shall return a list of ports associated with the given real_resource_id. The input parameter, port, is an identifier to a specific port, and the output parameter, port set, is a set of ports.

[7]Optional parameter.

--`,,`-`-`,,`,,`,`,,`---
Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

6.4.4.9.3 Completion status

Invalid port:	indicates that the port is not associated with this the resource.
Invalid real resource:	indicates that the real resource provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.4.4.10 [Real Resource] perform service

6.4.4.10.1 Service interface

Service:	<real_resource_id>.perform_service	
Input:	service id;	service_to_perform
Input:	behavior id;	behavior_id
Input:	at ports;	set_of_ports
Input:	parameters;	input_parameters
Output:[8]	measured_value;	data

6.4.4.10.2 Description

This service shall cause the behavior associated with a particular real resource to occur. The service id input parameter identifies which physical activity is to occur with the behavior, which is indicated by an identifier in the second input parameter. For example, the particular behavior for the real resource to exhibit is an ac signal at some frequency and power. The service id indicates whether the behavior is to be set up or applied. The parameter input values are the specific values for the behavior, such as a frequency of 100 MHz and a power of –10 dB. The one returned parameter, measured_value, is optional. This value represents returned data from the service, such as a measurement that was taken. In some cases, such as a stimulus, no value is returned.

6.4.4.10.3 Completion status

Invalid parameters:	indicates that the input data parameters are incorrect.
Invalid behavior:	indicates that the behavior_id is invalid.
Invalid service id:	indicates that the service_id parameter is invalid.
Invalid port:	indicates that an invalid port_id has been specified.
Invalid real resource:	indicates that the real resource provided is invalid or not created.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

[8]Optional parameter.

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

6.4.5 [Set of Real Resources]

6.4.5.1 State diagram

Figure 12 represents the state diagram for the set of real resources class.

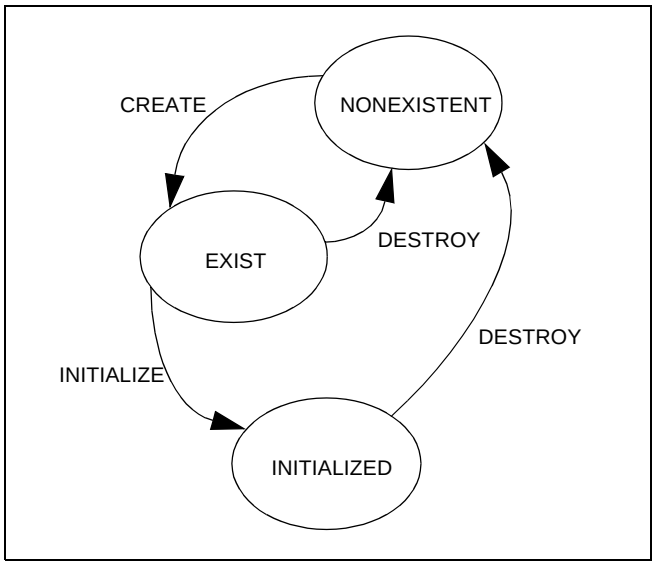

Figure 12—Set of real resources state diagram

6.4.5.2 [Set of Real Resources] create

6.4.5.2.1 6.4.5.2.1 Service interface

Service: create

Output: real resource set; real_resource_set_id

6.4.5.2.2 Description

This service creates an object of the set of real resources class. If the set of real resources cannot be created, a value that indicates this shall be returned.

Returned from this service is a set of real resources identifiers that are used to access the set.

6.4.5.2.3 Completion status

Unsuccessful operation: indicates that the service has failed.

6.4.5.3 [Set of Real Resources] initialize

6.4.5.3.1 Service interface

Service: <real_resource_set_id>.initialize

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Input:	name;	string
Input:	real resources;	set (real resource)

6.4.5.3.2 Description

This service shall initialize a previously created set of real resources object. The name parameter identifies a physical name associated with this set of real resources. The real resources parameter of type set (real resources) is created prior to calling this service. The set (real resources) inherits from the set class that is defined in the TFF collection module. This set class provides the following functions:

a) Add an item to a set.
b) Remove an item from the set.
c) Check if an item is a member of a set.
d) Retrieve the next item from the set.

A more detailed explanation of the set class can be found in the TFF description. Once initialized with a set (real resource) parameter, the set of real resources should not be changed, except through another call to initialize.

6.4.5.3.3 Completion status

Invalid real resource set:	indicates that the real resource set provided is invalid or not created.
Invalid name:	indicates that an illegal name has been used.
Invalid real resource:	indicates that the real resource identifier does not exist.
Invalid state:	indicates that the service cannot be performed while the object is in its current state.
Unsuccessful operation:	indicates that the service has failed.

6.4.5.4 [Set of Real Resources] destroy

6.4.5.4.1 Service interface

Service:	<real_resource_set_id>.destroy

6.4.5.4.2 Description

This service shall destroy an object of the set of real resources type by deleting its unique identifier. If the real resources in the set are still allocated, they will be deallocated through the deallocate service.

6.4.5.4.3 Completion status

Invalid real resource set:	indicates that the real resource set provided is invalid or not created.
Unsuccessful operation:	indicates that the service has failed.

--`,,`-`-`,,`,,`,`,,`---

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Annexes

Annex A

(informative)

Examples

This annex identifies examples of the usage of IEEE Std 1226.3-1998 services. The examples exist in written form and in graphical form. The main focus of the written examples is to indicate how some of the services could be utilized in various scenarios. The Clause 6 subclause numbers that apply to the services used are included in the example comments. Services that are used in these examples and defined in Clause 6 are highlighted by bold-face type.

A.1 Example 1—Virtual resource example

This example shows the possible implementation of a general ATLAS APPLY statement. The possible implementation is identified via a series of calls to the IEEE Std 1226.3-1998 services. Though this example implies that each of the service calls occurs in real-time, some of the services could actually be called at any time prior to runtime by any number of configuration tools.

The purpose of this APPLY statement is to cause the particular behavior, defined as an AC_SIGNAL with a certain POWER MAX and FREQUENCY, to occur at the P2-1 and P3-2 pins of the unit under test (UUT). The input to the P2-1 will be HI and the input to P3-2 will be LO. The connections are HI P2-1 and LO P3-2. A connection set is both of them together (i.e., HI P2-1 LO P3-2). HI and LO are undefined, but will eventually be linked to the HI and LO of the particular real resource that is used to supply the requested behavior.

The implementation of the APPLY will create a virtual resource with the requested behavior (AC SIGNAL, POWER MAX, FREQUENCY) and the connection set via the service map. In this case, the map function uses the virtual resource identifier from the USING parameter to retrieve the real resources and the path associated with the connection for the specified virtual resource. The returned real resource and path identifiers are then allocated through the IEEE Std 1226.3-1998 allocation services.

In an implementation in which SIG_GEN1 has not already been created, dynamic creation and allocation will occur. In this case, the map service becomes much more complicated. The following example defines an example map service implementation using dynamic creation and allocation.

```
012345 APPLY, AC SIGNAL USING 'SIG_GEN1', POWER MAX -5 DBM,
   FREQ 100 MHZ, CNX HI P2-1 LO P3-2 $

-- The creation of the behavior ids and the connection sets are shown in
-- Example 4 in Clause A.4.

-- This statement contains a USING parameter and an associated name, SIG_GEN1.
-- If no USING parameter is specified by the application, then the
-- implementation shall supply a generic name for internal use by the
-- initialization service (6.2.1).

sig_gen1_id ::= create
<sig_gen1_id>.initialize (name, behavior_id, connection_set_id)
```

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

```
<sig_gen1_id>.map

-- The following actions are a result of an example map implementation:

Begin (example of map)

  -- Within the map, search for candidate resources and candidate paths (6.3.1).
  behavior_id ::= <sig_gen1_id>.get_behavior
  resource_id_set ::= <test_equipment_configuration>.find_candidate_resources
        (behavior_id,non-allocated)

  -- Loop through this, look at the status and use some algorithm for choosing
  -- the best candidate resource(s). In this implementation, the most
  -- probable resources to use are placed at the front of the
  -- chosen_resource_id_set (6.4.4).

  loop on index
        candidate_resource_id ::= resource_id_set(index)
        resource_status ::= <candidate_resource_id>.get_status
        -- Perform choosing algorithm and create chosen_resource_id_set,
        -- which may or may not contain more than one real resource id.
  end loop

  -- In the rare case in which no candidate resources exist, the mapping service
  -- shall end at this point and return the appropriate error status that
  -- indicates unsuccessful completion. Otherwise, it will continue on by
  -- getting the set of candidate paths associated with each candidate
  -- resource in the chosen_resource_id_set (6.2.1, 6.3.1).

  loop on index
        candidate_resource_id ::= chosen_resource_id_set(index)
        connection_set_id ::= <sig_gen1_id>.get_connection_set
        set_of_path_sets ::= <test_equipment_configuration>.
              find_candidate_paths (candidate_resource_id, connection_set_id)
        -- A check is made to see if candidate paths exist for this resource.
        -- If not, the resource is removed from the candidate set.
  end loop on index

  -- Some algorithm is performed to choose the best resource and path,
  -- if more than one candidate resource and associated paths exist, and
  -- to determine whether or not the candidate path is available.

  -- Now that a resource and associated path(s) have been found, they must be
  -- allocated, with the resource being allocated first (6.4.1, 6.4.4).

  <chosen_resource_id>.allocate

  -- Now, allocate the chosen path(s).

  if more than one path then
        loop on index
              <chosen_path_id_set(index)>.allocate
        end loop
  else if one path then
        <chosen_path_id_set(1)>.allocate
  else
        no path is available
  end if more than one path
end Map implementation
```


Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

```
-- Cause the behavior of the created virtual resource to occur (6.2.1).

<sig_gen1_id>.perform_service(apply, parameters)

  -- Physical calls to control the instruments and path.
```

A.2 Example 2—Virtual resource example with pre-execution tools applied

This example shows the possible implementation of a general ATLAS APPLY statement, as described in the previous example, except that the virtual resources have been predefined. An example of how the virtual resources could be predefined can be found in Example 8 in Clause A.8.

As in Example 1 in Clause A.1, the purpose of this APPLY statement is to cause the particular behavior, defined as an AC_SIGNAL with a certain POWER MAX and FREQUENCY, to occur at the P2-1 and P3-2 pins of the UUT. The input to the P2-1 will be HI and the input to P3-2 will be LO. The connections are HI P2-1 and LO P3-2. A connection set is both of them together (i.e., HI P2-1 LO P3-2). HI and LO are undefined, but will eventually be linked to the HI and LO of the particular real resource that is used to supply the requested behavior.

The implementation of the APPLY will create a virtual resource with the requested behavior (AC SIGNAL, POWER MAX, FREQUENCY) and the connection set via the service map. In an implementation in which SIG_GEN1 has already been created, real resources have already been mapped, and paths to the real resources have already been created, the virtual resource map service is very simple. The map function uses the virtual resource identifier from the USING parameter to retrieve the real resources and the path associated with the connection for the specified virtual resource. The returned real resource and path identifiers are then allocated through the IEEE Std 1226.3-1998 allocation services. The following example defines an example map service implementation using the predefined creation of virtual resources.

```
012345 APPLY, AC SIGNAL USING 'SIG_GEN1', POWER MAX -5 DBM, FREQ 100 MHZ,
       CNX HI P2-1 LO P3-2 $

-- The creation of the behavior ids and the connection sets are shown in
-- Example 4 in Clause A.4.

-- This statement requires a USING parameter and an associated name, SIG_GEN1.
-- This name is used by the implementation to retrieve the previously created
-- virtual resource information.

-- No create or initialization of the virtual resource is performed, since it
-- is assumed this has already been done, with sig_gen1_id being the
-- identifier for the object named SIG_GEN1.
-- See Example 8 in Clause A.8 for how this could be accomplished with a tool.

-- A sample implementation of the APPLY may do the following (6.2.1):

<sig_gen1_id>.map

--
-- Implementation of map based on the use of pre-execution tools:
Begin (example of map)

  -- Retrieve the predefined mapping of the virtual resource (6.2.1).

  mapping_ID ::= < sig_gen1_id >.get_mapping
```

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

```
  -- The mapping_ID has two pieces, the resource list and the path list.

  -- The resource associated with the mapping_ID.resource is allocated (6.4.4).

  <mapping_ID.resource_ID>.allocate

  -- Next, the path associated with the mapping_ID.path is allocated
  -- and the connection is established (6.4.1).


   <mapping_ID.path_ID>.allocate

end (Map implementation)

-- After the map service is completed, the APPLY will cause the behavior of
-- the virtual resource to occur (6.2.1).

<sig_gen1_id>.perform_service(apply, parameters)

  -- Physical calls to control the instruments and path.
```

A.3 Example 3—Virtual resource group example

This example shows the possible implementation of two ATLAS APPLY statements whose resources must be made available concurrently. This is implemented by using the virtual resource group. The possible implementation of a virtual resource group is identified via a series of calls to the IEEE Std 1226.3-1998 services.

```
012340  DO, SIMULTANEOUS $
012345  APPLY, AC SIGNAL USING 'SIG_GEN1', POWER MAX -5 DBM, FREQ 100 MHZ,
        CNX HI P3-3 LO P1-3 $
012350  APPLY, AC SIGNAL USING 'SIG_GEN2', POWER MAX 0 DBM, FREQ 300 MHZ,
        CNX HI P2-1 LO P3-2 $
012355  END, DO $

-- Create the virtual group (group1) and two virtual resources (sig_gen1 and
-- sig_gen2). The creation of the behavior ids and the connection sets are
-- shown in Example 4 in Clause A.4.

-- These statements contain USING parameters and associated names, SIG_GEN1
-- and SIG_GEN2.
-- If no USING parameter is specified by the application, then the
-- implementation will supply a generic name for internal use by the
-- initialization service. (6.2.5)

group1_id ::= create
<group1_id>. initialize (group1)
sig_gen1_id ::= create
<sig_gen1_id>. initialize ('SIG_GEN1', behavior_id, connection_set_id)
sig_gen2_id ::= create
<sig_gen2_id>. initialize ('SIG_GEN2', behavior_id, connection_set_id)

-- Add the newly created virtual resources to the virtual resource group,
-- which, in this implementation, adds each virtual resource id to the
-- virtual_id_set structure accessed by the group1_id.
```


Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

```
<group1_id>.add_virtual_resource (sig_gen1_id)
  -- Bump virtual_id_set by one and add sig_gen1_id to the structure.

<group1_id>.add_virtual_resource (sig_gen2_id)
  -- Bump virtual_id_set by one and add sig_gen2_id to the structure.

-- Map each of the virtual resources to physical resources via a map command
-- to the group (6.2.1 and 6.2.5).

<group1_id>.map

Begin -- map implementation

  -- The first task of the group map command is to loop through each virtual
  -- resource to find candidate resources and paths. If the process fails to
  -- find a map, then no resources or paths are allocated, and the service
  -- returns with an appropriate error that indicates unsuccessful completion.

  loop through multiple virtual_id_set structures within group1_id using
  index1

        -- Find candidate resources for each of the virtual resources in
        -- group1_id (6.2.1, 6.3.1).

        behavior_id ::= <virtual_id_set(index1))>.get_behavior
        resource_id_set(index1,starting_index) ::=
              <test_equipment_configuration>.
              find_candidate_resources(behavior_id,All_Resources)

        -- A search is made through the resource_id_set. If any of the
        -- virtual resources were unable to be mapped to real resources, then
        -- the group map service will exit and return an unsuccessful
        -- completion status.

        -- If possible real resources were found for all virtual resources in
        -- group1_id, then a loop through the resource_id_set created is made
        -- to search for possible paths for each of the possible resources
        -- (6.2.1, 6.3.1).

        loop through multiple resource_id_set structures within group1_id
         using index2
              connection_set_id := <virtual_id_set(index1))>
              .get_connection_set
              path_id_set(index1,index2) ::= <test_equipment_configuration>.
                    find_candidate_paths (resource_id_set(index1,index2),
                                          connection_set_id)
        end loop through multiple resource_id_set structures within group1_id

  end loop through multiple virtual_id_set structures within group1_id

  -- A search is made through the path_id_set. If any of the possible
  -- resources were unable to be associated with paths, then the possible
  -- resources are removed from the resource_id_set. When finished, a set of
  -- possible resources and associated possible paths exists. A loop is
  -- performed that analyzes this set and creates a chosen_resource_set and
  -- a chosen_path_set. Part of the analysis includes calls to get_status
  -- for each of the possible resources and paths (6.4.1, 6.4.4).

  loop through resource_id_set and path_id_set using index1,index2
```

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

```
        resource_status ::= <resource_id_set(index1,index2)>.get_status
        path_status ::= <path_id_set(index1,index2)>.get_status
        -- Perform further analysis to create chosen_resource_set and
        -- chosen_path_set.
  end loop through resource_id_set and path_id_set

        -- Now that a path(s) and resource(s) have been found, they must be
        -- assigned to the virtual resources. First, assign the mapping of
        -- the resource(s) (6.2.1).

  loop through multiple virtual_id_set structures within group1_id using
   index1


   <virtual_id_set(index1)>.assign_mapping(chosen_resource_set(index1,1),
                    chosen_path_set(index1,1))
          Begin assign_mapping implementation for virtual resource 1

              -- The main task of assign mapping is to allocate the real
              -- resources and paths. Some mapping and storage of mapping
              -- must also occur, but is not included in this example.

              -- First, allocate the first resource in the set (6.4.4).

              <chosen_resource_set (index1,1)>.allocate

              -- Now, allocate the chosen path(s) of the first virtual
              -- resource.

              if more than one path then

                     loop on index2
                            <chosen_path_set(index1,index2)>.allocate
                     end loop
              else if one path then
                     <chosen_path_set(index1,1)>.allocate
              else
                     no path is available
              end if more than one path

          end assign_mapping implementation for virtual resource 1

  end loop through multiple virtual_id_set structures within group1_id

end Map implementation

-- Cause the behavior of the created virtual resources to occur. The
-- simultaneity of the APPLY statements in this virtual resource group example
-- is performed based on the implementation of the perform_service service.
-- This example, and the purpose of the virtual resource group, is only to
-- assure the user that real resources exist that will allow the multiple
-- apply statements to be implemented at the same time (6.2.1).

<sig_gen1_id>.perform_service(apply, parameters)
<sig_gen2_id>.perform_service(apply, parameters)
```


Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

A.4 Example 4—Behavior and connection creation

This example utilizes the same ATLAS statement as utilized in Example 1 in Clause A.1. This ATLAS statement is as follows:

```
012345  APPLY, AC SIGNAL USING 'SIG_GEN1', POWER MAX -5 DBM, FREQ 100 MHZ,
        CNX HI P2-1 LO P3-2$

-- This series of creations could occur at execution time or at some time well
-- before execution time.

-- First, the behavior object, defined in the TFF, is created from the signal
-- information in the ATLAS statement

behavior_1::= create
<behavior_1>.initialize ( "Behavior 1", ( Power, max -5 dbm, Freq, 100 Mhz) )

-- Next, the connection (6.2.2) and connection sets (6.2.3) are created.

connection_hi_id::= create
<connection_hi_id>.initialize ( CON_HI, HI, P2-1 )

connection_lo_id::= create
<connection_lo_id>.initialize ( CON_LO, LO, P3-2 )

-- The connection set is created and the connections are added to the
-- set of connections that is an object of type Connection_UObset,
-- which is defined in the TFF.

connection_set_id::= create
<connection_data_id>.add (connection_lo_id)
<connection_data_id>.add (connection_hi_id)

<connection_set_id>.initialize(connection_data_id)

-- Finally, the actual signal object is created. This object matches the
-- object created in Example 1 in Clause A.1.

-- This statement contains a USING parameter and an associated name, SIG_GEN1.
-- If no USING parameter is specified by the application, then the
-- implementation will supply a generic name for internal use by the
-- initialization service (6.2.1).

sig_gen1_id::= create
<sig_gen1_id>.initialize ('SIG_GEN1', behavior_1, connection_set_id)

-- Once created, the signal object can be mapped to real resources.
```

A.5 Example 5—Dynamic selection using real resource example

This example shows the possible implementation of a general ATLAS APPLY statement without the use of a virtual resource (i.e., subset 3 of Table 1). Though the code shown is an example implementation of the APPLY statement, the code itself could be made part of the test program instead of the ATLAS statement. In other words, the test program could be written using direct calls to the IEEE Std 1226.3-1998 services and still be portable across implementations.

--`,,`-`-`,,`,,`,`,,`---
Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

The possible implementation of this APPLY is identified via a series of calls to the IEEE Std 1226.3-1998 test equipment configuration and real resource services. This example uses the services in the test equipment configuration to search for paths and resources to meet the needs of the APPLY. This example can be found pictorially in Annex B, Figure B.5.

```
012345  APPLY, AC SIGNAL USING 'SIG_GEN1', POWER MAX -5 DBM,FREQ 100 MHZ,
        CNX HI P2-1 LO P3-2$

Begin (example of APPLY implementation)

  -- Get the behavior id (6.4.4) and search for candidate resources (6.3.1)
  -- through a call to find_candidate_resources.

  behavior_id ::= <sig_gen1_id>.get_behaviors
  resource_id_set ::= <test_equipment_configuration>.find_candidate_resources
       (behavior_id,non-allocated)

  -- Loop through the resource_id_set that was found. Look at the status and use
  -- some algorithm for choosing the best candidate resource(s). In this
  -- implementation, the most probable resources to use are placed at the
  -- front of the chosen_resource_id_set (6.4.4).

  loop on index
       candidate_resource_id ::= resource_id_set(index)
       resource_status ::= <candidate_resource_id>.get_status
       -- Perform choosing algorithm and create chosen_resource_id_set,
       -- which may or may not contain more than one real resource id.
  end loop

  -- In the rare case in which no candidate resources exist, the APPLY service
  -- will end at this point and return the appropriate error status, thus
  -- indicating unsuccessful completion. Otherwise, it will continue on by
  -- getting the set of candidate paths associated with each candidate
  -- resource in the chosen_resource_id_set. The connection set has
  -- previously been created and initialized.

  loop on index
       candidate_resource_id ::= chosen_resource_id_set(index)
       set_of_path_sets ::= <test_equipment_configuration>.
             find_candidate_paths (candidate_resource_id,
                   connection_set_id)
       -- A check is made to see if candidate paths exist for this resource.
       -- If not, the resource is removed from the candidate set.
  end loop on index

  -- Some algorithm is performed to choose the best resource and path, if
  -- more than one candidate resource and associated paths exist, and
  -- to determine whether or not the candidate path is available.

  -- Now that a resource and associated path(s) have been found, they must be
  -- allocated, with the resource being allocated first.

  <chosen_resource_id>.allocate

  -- Now, allocate the chosen path(s).

  if more than one path then
       loop on index
```


Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

```
            <chosen_path_id_set(index)>.allocate
      end loop
   else if one path then
      <chosen_path_id_set(1)>.allocate
  else
      no path is available
  end if more than one path
  ports_id ::= <chosen_resource_id>.get_ports

  -- Ports associated with the ports_id and connections associated with
  -- the connection_set_id are analyzed. In this example, the HI and LO
  -- ports on the chosen resource are identified. Connect (6.4.1) is called
  -- to set up the path. A call to perform_service (6.4.4) is then made to
  -- start the control of the resource.

  if more than one path then
      loop on index
            <chosen_path_id_set(index)>.connect
      end loop
  else if one path then
      <chosen_path_id_set(1)>.connect

  <chosen_resource_id>.perform_service ( apply, behavior_id, ports_id )


end APPLY implementation
```

A.6 Example 6—Manual selection using real resource example

This example shows the possible implementation of a general ATLAS APPLY statement without the use of a virtual resource. The possible implementation is identified via a series of calls to the IEEE Std 1226.3-1998 real resource services. This assumes that the programmer knows the real resources in the tester (such as SIG_GEN1) and the associated paths. This example can be found pictorially in Annex B, Figure B.6.

```
012345  APPLY, AC SIGNAL USING 'SIG_GEN1', POWER MAX -5 DBM,FREQ 100 MHZ,
        CNX HI P2-1 LO P3-2$

-- Assuming that there is a table associating the using value with a real
-- resource id of real_sig_gen1 (6.4.4).

Status_of_real_sig_gen1 := <real_sig_gen1>.get_status;

if Status_of_real_sig_gen1 = Available then
  <real_sig_gen1>.allocate;

  -- Assuming that there is a table associating the terminal/pin to a path
  -- of hi_path,lo_path (6.4.4).

  Status_of_hi_path := <hi_path>.get_status;
  Status_of_lo_path := <lo_path>.get_status;

  if Status_of_hi_path and Status_of_lo_path = Available then
      <hi_path>.allocate;
      <lo_path>.allocate;

      ports_id ::= < real_sig_gen1>.get_ports
```

--`,,`-`-`,,`,,`,`,,`---
Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

```
        -- Ports associated with the ports_id and connections associated with
        -- the connection_set_id are analyzed. In this example, the HI and
        -- LO ports on the chosen resource are identified. Connect is called
        -- to set up the path and a call to perform_service is then made to
        -- start the control of the resource (6.4.4).

        <hi_path>.connect;
        <lo_path>.connect;
        < real_sig_gen1>.perform_service ( apply, behavior_id, ports_id )
  else
        <real_sig_gen1>.deallocate;
        -- Exit, you can not continue.
  endif
else
        -- Exit, you cannot continue.
endif
```

A.7 Example 7—Non-signal-oriented example

Non-signal-oriented statements can also be represented by the IEEE Std 1226.3-1998 services. Test program sets (TPS) can be written in Ada, C/C++, FORTRAN, or many other languages. For example, an ATLAS measure can be represented as follows:

```
MEASURE, (VOLTAGE INTO RESULT), DC SIGNAL USING DMM1-DC-VOLTS, VOLTAGE MAX
100V,         CNX HI J1-8 LO J1-21$
```

This statement can be represented with the following non-signal-oriented statements:

```
Configure_Input (DMM1, DC_VOLTS, 100V);
DMM1HI_LO.Connect;
Read_Measurement (DMM1, RESULT);
DMM1HI_LO.Disconnect;
```

The configure and read statements are operations that are compliant with the VXI plug&play System Alliance drivers, and are also compliant with the IEEE P1226.4/D2.1 draft standard. The connect and disconnect operate on the predefined signal paths in the connection set, DMM1HI_LO. These paths have been set up and stored in a flat file, database, or some other proprietary method at some time before test program execution. These paths and connections can be created using the following IEEE Std 1226.3-1998 services (see 6.2.2 and 6.2.3):

```
  DMM1HI_J1-8 ::= create;
  <DMM1HI_J1-8>.initialize( 'HI_CNX', DMM1HI, J1-8);

  DMM1LO_J1-21 ::= create;
  <DMM1LO_J1-21>.initialize ( 'LO_CNX', DMM1LO, J1-21);

  DMM1HI_LO ::= connection_set.create;
  DMM1HI_LO.initialize ( DMM1HI_J1-8, DMM1LO_J1-21);
```

The utilization of these connection set routines and the four non-signal-oriented statements are completely portable to any systems that have a compliant IEEE Std 1226.3-1998 and IEEE P1226.4/D2.1 interface. The implementation of the four services used to mimic the ATLAS statement and the initialization service calls to create the paths and connection set could be written in any language. Additionally, these services could be built into graphic front panel utilities and created with commercial test program development tools. No mat-

--`,,`-`-`,,`,,`,`,,`---

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

ter how the software is created, the services are portable to other systems that have an IEEE Std 1226.3-1998 and IEEE P1226.4/D2.1 service interface implementation.

A.8 Example 8—Pre-execution virtual resource tool

This tool creates a series of virtual resource names at pre-execution time that are usable by the test programmer at execution time. Figure A.1 identifies how this tool could work by using the high level services of the IEEE Std 1226.3-1998 standard and the low-level services defined in the TRIM. The virtual resource information is generated by the tool and creates a mapping of these virtual resource names to actual real resources and paths. This mapping can be in a database, flat file, or whatever the implementation dictates. If each of the virtual resources needs to be simultaneously available, the tool can be modified to use the virtual resource group management services. This will cause unique real resources and paths to be mapped to each virtual resource.

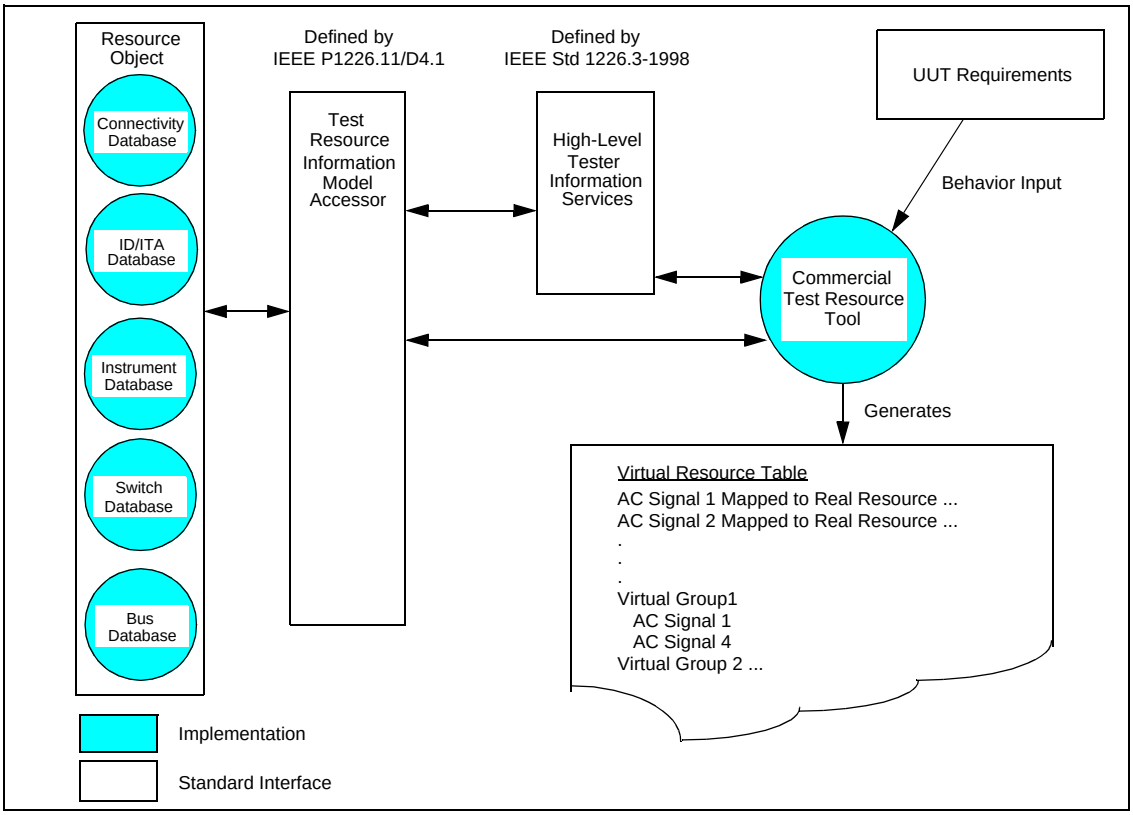

Figure A.1—Pre-execution virtual resource generation example

During execution, the test executive could use the virtual resource table for rapid access to information about the mapping from virtual to real. In addition to the high-level functions, Figure A.1 identifies the utilization of low-level services. This is an optional interface, but the tool may need to access control or bus information to be included in the mapping table. If this is done, applications during execution have ready access to this information through the virtual resource.

The following is an example flow of the pre-execution tool:

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

```
-- Create the virtual resource identifiers to be utilized (6.2.1).

virtual_resource_array(1) ::= create;
virtual_resource_array(2) ::= create;
virtual_resource_array(3) ::= create;
virtual_resource_array... (continue creation of other virtual resources)

-- Initialize the virtual resource sets previously created.

virtual_resource_array(1).initialize(virt_res1, behavior_id1, connect_set1);
virtual_resource_array(2).initialize(virt_res2, behavior_id2, connect_set2);
virtual_resource_array(3).initialize(virt_res3, behavior_id3, connect_set3);
virtual_resource_array(4).initialize... (continue initialization of other vir-
tual resources)

-- Search for candidate resources and candidate paths for each virtual resource.

-- Get the set of candidate resources (6.3.1).

loop on set of virtual resources using index1
  behavior_id ::= <virtual_resource_array(index1)>.get behavior
  resource_id_set ::= <test_equipment_configuration>.find_candidate_resources
       (behavior_id,all)

  -- Loop through the set of resources and find the candidate paths
  -- (6.2.1, 6.3.1).

  loop on index2
       connection_set_id ::= <virtual_resource_array(index1)>.get_connection_set
       path_id_set(index1,index2) ::= <test_equipment_configuration>.
              find_candidate_paths (resource_id_set(index2), connect_set_id)

       -- Loop through this set and use some algorithm for choosing
       -- the best candidate path(s) and for placing them in chosen_path_set
       -- along with a chosen_resource_set.

  end loop on index2

end loop on set of virtual resources using index1

loop on set of virtual resources using index1

  -- The virtual to physical mapping could be stored in a database, flat
  -- file, or some other storage facility. This example, however, does not
  -- explicitly show this storage.

  -- Finally, the virtual instrument is mapped to the real resources (6.2.1).

  <virtual_resource_array(index1)>.assign_mapping(chosen_resource_set(index1),
              chosen_path_set(index1))

end loop on set of virtual resources using index1
```


--`,,`-`-`,,`,,`,`,,`---
Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

A.9 Example 9—UUT/tester analysis tool

This tool will perform an analysis of any tester to see if it is able to test a particular UUT. The key to the analysis is based on matching the behaviors of the UUT that need to be tested to the behaviors that can be tested by the tester. Figure A.2 identifies the process and use of IEEE Std 1226.3-1998.

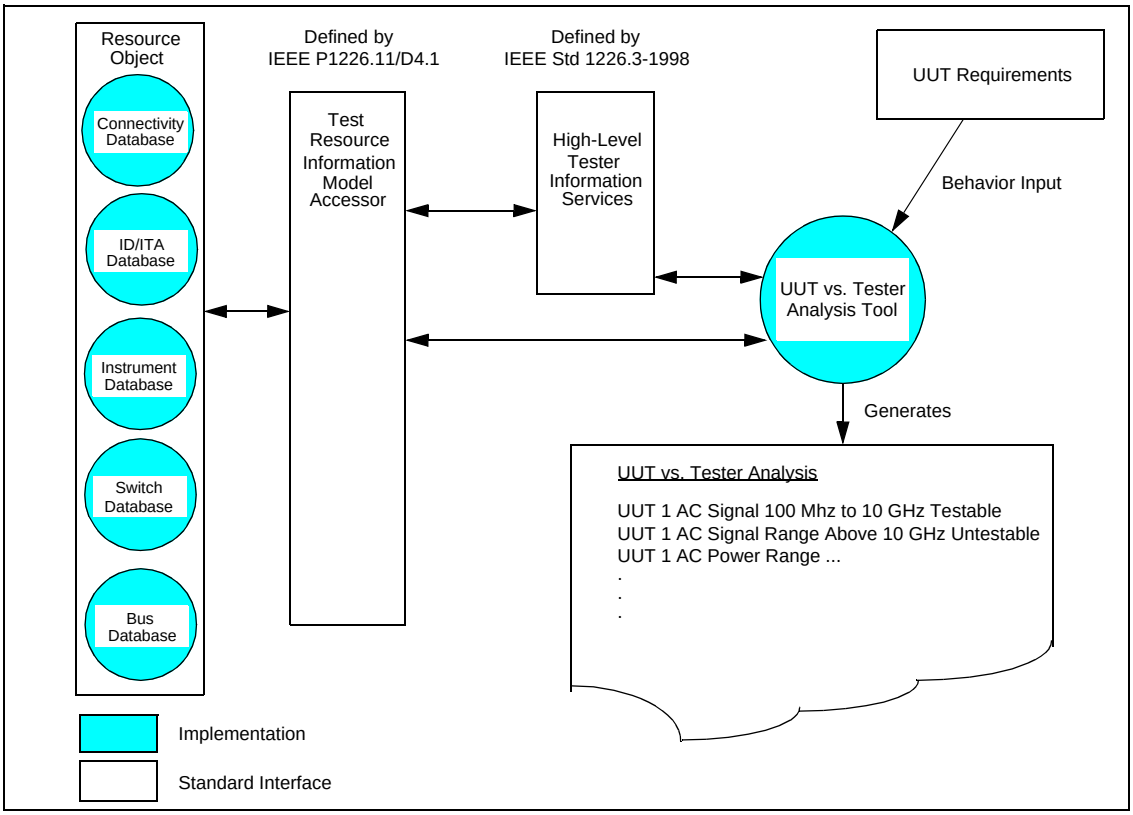

Figure A.2—UUT/tester analysis tool example

```
-- Each of the IDs created can represent a group of test equipment, such as an
-- IFTE or a CASS (6.3.1).

tec_id_set(1) ::= create
tec_id_set(2) ::= create
tec_id_set(3) ... continue creation of test equipment configurations

-- Search TEC1 for candidate resources and paths. If no IDs are returned,
-- then TEC2 and subsequent test equipment configurations are searched. The
-- behavior used in the search is the behavior required to test the UUT.

loop through test equipment configurations on index1

  real_id_set := <tec_id_set(index1)>.find_candidate_resources
       (behavior_id,all)
  path_id_set := <tec_id_set(index1)>.find_candidate_paths
       (real_id_set,connect_set)
  If real_id_set > 0 and path_id_set > 0 then
       exit loop
```

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

```
  else
       continue loop
  end if
end loop

-- If the real_id_set and path_id_set are still empty at the end of this
-- loop, then no configuration known to this tool can test the given UUT
-- requirement.
```

A.10 Example 10—Sample ABBET lower layers architecture

Figure A.3 identifies an example of the ABBET lower layers architecture. Many other examples exist with respect to IEEE Std 1226.3-1998, but this one focuses on a possible interrelation between IEEE Std 1226.3-1998, IEEE P1226.4/D2.1, and IEEE P1226.5. The shaded circles and boxes identify implementations of the ABBET standards. A particular application is shown in the clear circle. The application could be an ATLAS program or some other type of program, including commercial graphical environments. A typical test system contains most of the information indicated in the five shaded circles on the left side of the figure. Other information may exist, but these five were used as an example. An ABBET compliant test system must utilize standard services to access the information indicated in these shaded circles. As the diagram indicates, these services are the low-level tester information services defined in the TRIM.

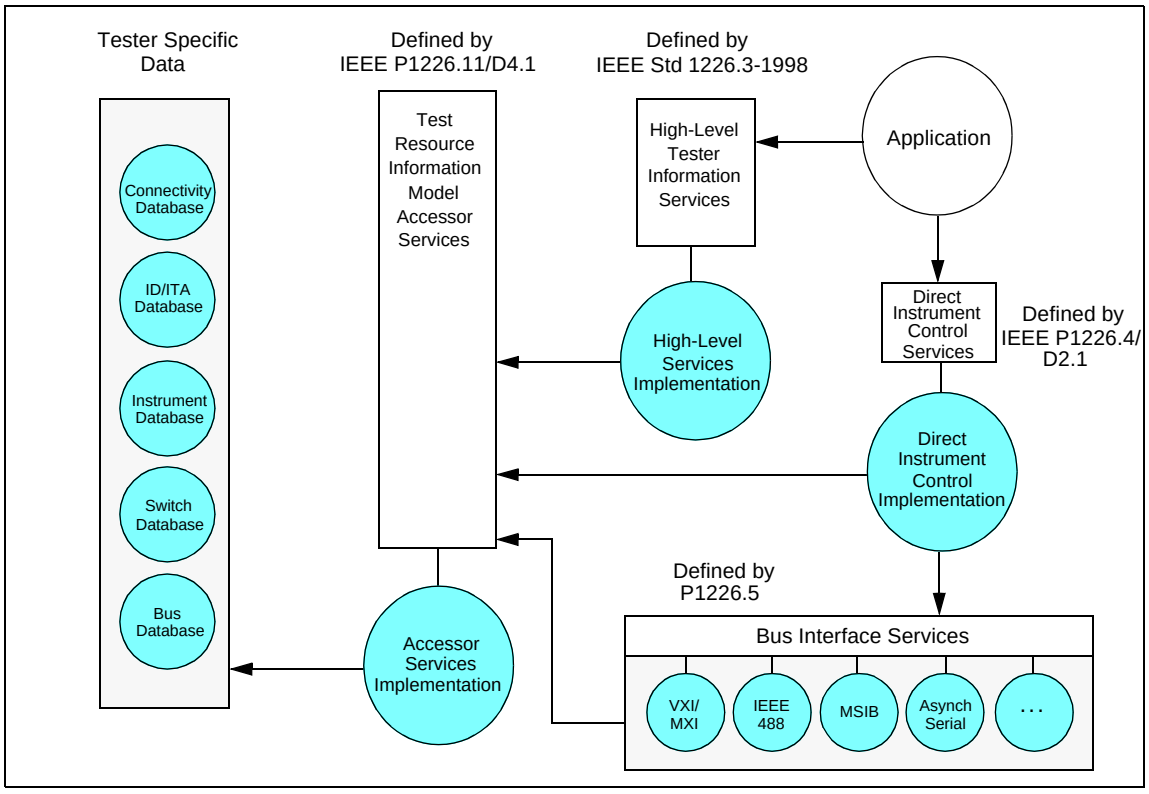

Figure A.3—Example of ABBET lower layers architecture

IEEE draft standards IEEE P1226.4/D2.1 and IEEE P1226.5 are also shown in Figure A.3. Since they do not pertain directly to IEEE Std 1226.3-1998, they will not be discussed in detail. However, it is evident from

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Figure A.3 that the application calls the direct instrument control services in IEEE P1226.4/D2.1, which in turn call the bus interface services in IEEE P1226.5 to control and utilize the instruments and buses.

This example is only one method of using IEEE Std 1226.3-1998, IEEE P1226.4/D2.1, and IEEE P1226.5. The scenario discussed, for instance, involves most of the services being called at execution time. It is possible, and most probable, however, that tools specific to various areas of the process will be developed to enhance the execution time environment. Some of these tools have been shown in the previous examples.

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

Annex B

(informative)

Object-oriented analysis

The purpose of this annex is to identify the rationale and methodology for the development of the IEEE Std 1226.3-1998 objects, classes, and related services.

In order to define the architecture of the interfaces at the test resource level, an object-oriented analysis (OOA) methodology was chosen for the design process. OOA is one method for analyzing and defining complex system architectures.

The complexity of the ABBET test resource management architecture has been made more manageable through the use of the OOA methodology. OOA provides a view of a complex system via the embodiment of its elements into objects and classes. In addition, OOA describes the attributes, abstractions, encapsulation, modularity, hierarchy, typing, concurrence, and persistence of the objects and classes.

Specifically, OOA has been used in the development of the test resource management architecture to identify the classes and objects, their relationships, and, ultimately, the messages that define the transfer of information and data. The sets of services are defined based on the messages. They are partitioned into IEEE Std 1226.3-1998 and IEEE P1226.4/D2.1, where they are expressed in a language-independent specification format. Language bindings are defined in the standard to support the use of the services by test programs. The language bindings are defined using IDL, which is defined by the Object Management Group (OMG).

OOA methods have been developed by a number of people including Booch, Rumbaugh, and Coad/Yourdon. Each author, however, utilizes a different notation. An investigation determined that all methods include the basic concepts of classes, objects, and relationships. Additionally, the investigation concluded that using a particular OOA method and notation would not preclude the conversion to a different method and notation at a later date, if needed. With these caveats, the Booch OOA methodology [B1][9] was chosen to be used for the definition of the IEEE ABBET test resource OOA.

B.1 Resource management class model

This standard focuses on the resource management class model shown in Figure B.1. The top-most class is resource, and this class corresponds directly with the resource class found in the ABBET model in IEEE Std 1226-1993. The classes in Figure B.1 are shown as dotted line “clouds,” with the relationships indicated by lines between the clouds. The underlined class names indicate the classes that are defined in the TFF, which may be found in IEEE Std 1226-1993, or in the TRIM, which is a proposed component standard of IEEE Std 1226-1993. The definitions for the classes in Figure B.1 are as follows:

a) *Resource:* An abstract superclass of objects in the test station. The only purpose of this class is to collect relationships and to hold data structures that are shared by its subclasses. The subclasses of resource are real resource element, test equipment configuration, and virtual resource. In addition, the following subclasses of these subclasses are: set of real resources, real resource, individual resource, and composite resource. There are no instances of this class, there are only instances of its nonabstract subclasses.

[9]The numbers in brackets preceded by the letter B correspond to those of the bibliography in Annex D.

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

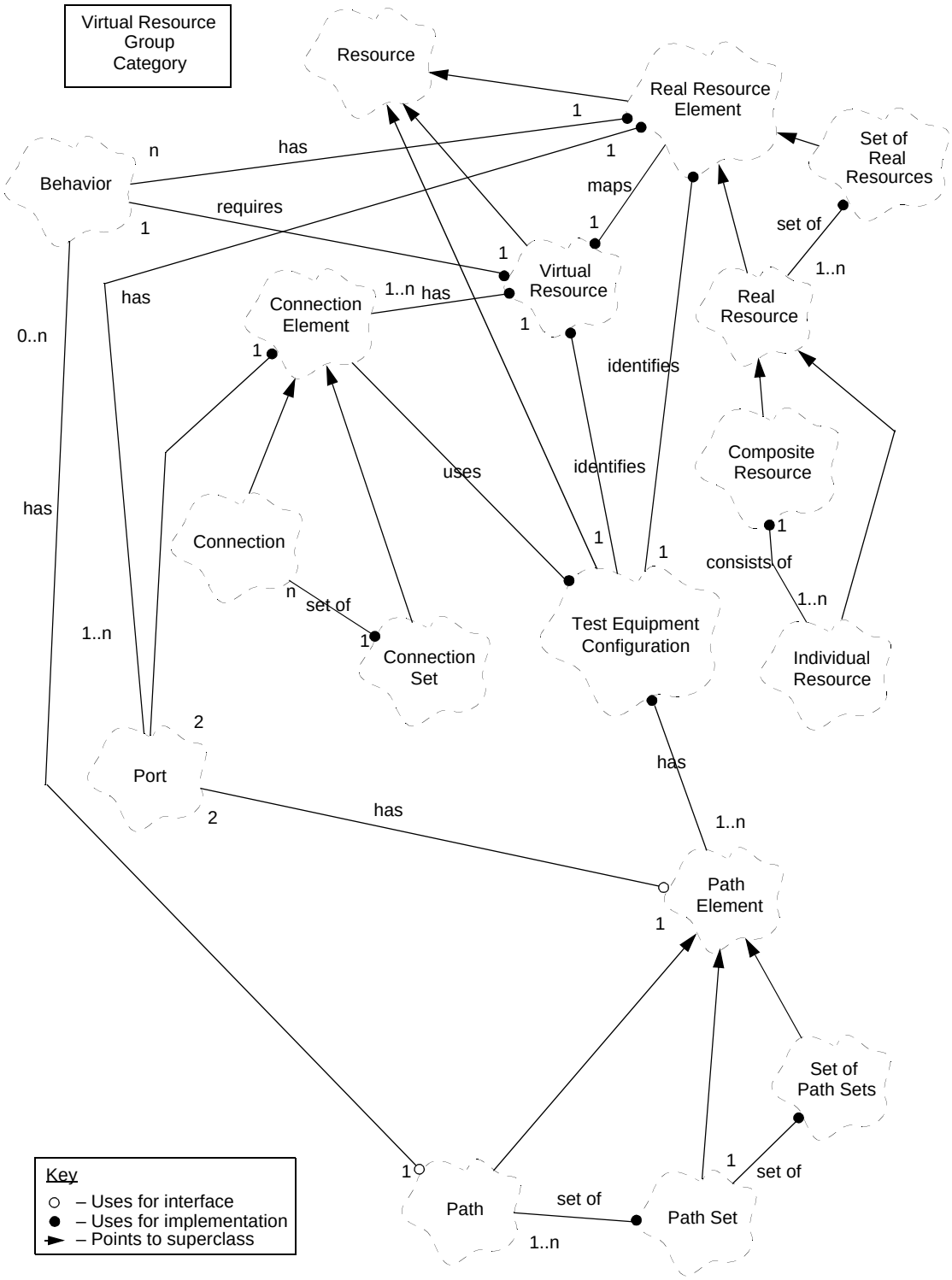

Figure B.1—Resource management class model

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

b) *Individual resource:* A class of objects that represents individual, physical instruments. Each individual resource object has a set of behavior descriptions that is indicated in Figure B.1 through an inheritance to real resource, real resource element, and resource. It is the resource class that shows the actual relationship to behavior. These physical instruments are the actual equipment in a test system, such as signal generators, power supplies, and switches. This class is a subclass of the real resource class.

c) *Real resource element:* An abstract class of objects that is a subclass of resource. There are no instances of this class. Its only purpose is to collect relationships and to hold data structures that are shared by its two subclasses, but are not shared by its parent class. The two subclasses are set of real resources class and real resource class.

d) *Real resource:* An abstract class of objects that has two subclasses. Its only purpose is to collect relationships and to hold data structures that are shared by its two subclasses, but are not shared by its parent class. The two subclasses of real resource are composite resource and individual resource. No actual objects of this class will exist. Only objects of the two subclasses exist.

e) *Set of real resources:* A class of objects that represents a set of abstract classes of objects identified as real resources. A set of real resources has the same characteristics as the real resources class previously discussed.

f) *Virtual resource:* A class of objects that represents virtual resources as defined by the IEEE ABBET standards. Each virtual resource is a description of a required capability needed to test a function of a test subject. Virtual resource is a subclass of resource. It maps to a real resource element and has a connection element. The connection, properties, and mapping are identified in the test equipment configuration.

g) *Composite resource:* A class of objects that represents one or more capabilities that are represented by more than one physical or individual resource. Each composite resource object has a set of behavior descriptions that describe the behaviors provided by the resource. This class is a subclass of the real resource class.

h) *Test equipment configuration:* A class of objects that represents a configuration of real resources. This class would have only one instance in the implementation of one automatic test station, but would have multiple instances in a data base of all test stations in inventory. This class is a subclass of the resource class.

i) *Path element:* An abstract class of objects. There are no instances of this class. Its only purpose is to collect relationships and to hold data structures that are shared by its three subclasses. The three subclasses are path, path set, and set of path sets. This abstract class is not defined in Clause 6 of this standard. The subclass definitions contain all of the attributes and methods.

j) *Path:* A class of objects that represent a path through the test system. These paths route resources to UUT ports. The relationship to ports and the test configuration is shown through the relationship between the port class and test equipment configuration class with the path element class, which is the parent class of path.

k) *Path set:* A class of objects that represents a set of paths through the test system. The relationship to ports and the test configuration is shown through the relationship between the port class and test equipment configuration class with the path element class, which is the parent class of path set.

l) *Set of Path set:* A class of objects that represents a set of path sets through the test system. The relationship to ports and the test configuration is shown through the relationship between the port class and test equipment configuration class with the path element class, which is the parent class of path set.

m) *Port:* A class of objects that represents connection points on physical resources, paths, and UUTs. The relationship to other classes in the resource management model includes real resource element, path element, and connection element.

n) *Behavior:* A class of objects that represents the set of behavioral parameters of a resource or a path. A capability can include frequency ranges, voltage levels, and other associated descriptors.

o) *Connection element:* An abstract class of objects. There are no instances of this class. Its only purpose is to collect relationships and to hold data structures that are shared by its subclasses. The subclasses are connection and connection set. This abstract class is not defined in Clause 6 of this standard. The sub-class definitions contain all of the attributes and methods.

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

p) *Connection:* A class of objects that represents the virtual route between the test subject pin and the test equipment configuration input port. The relationship to ports, virtual resources, and their configuration is shown through the relationship between the port class, virtual resource class, and test equipment configuration class with the connection element class, which is the parent class of connection.

q) *Connection set:* A class of objects that represents a set of connections. A connection is the virtual route between the unit under test pin and the test equipment configuration input port. The relationship to ports, virtual resources, and their configuration is shown through the relationship between the port class, virtual resource class, and test equipment configuration class with the connection element class, which is the parent class of connection set.

Relationships between classes in Figure B.1 are shown by a line between the classes. The type of line specifies the type of relationship. A line with an open circle between classes indicates that the class at the end of the line with the circle (the using class) “uses for interface” the class at the other end of the connection (the used class). This documents that the using class requires objects of that class or data defined in the class for the interface to one or more of the using class’s methods. A filled circle indicates that the class at the end of the line with the circle “‘uses for implementation” the other class. This indicates that the using class requires objects of the class or data defined in the class for the implementation of the using class’s methods, but that these items are not visible at the interface of the using class’s methods. For example, objects of the path element class use an object of the port class as data (as a parameter to a method), while the objects in the test equipment configuration class actually call services defined in the path element class to implement a path element class method.

The lines with arrow heads indicate a subclass relationship, with the arrow head specifying the superclass. For example, the classes composite resource and individual resource are subclasses of real resource, which, in turn, is a subclass of real resource element, which is a subclass of resource. Therefore, composite resource and individual resource inherit the relationships of real resource class to set of real resource and real resource element to port element and behavior. However, subclasses can have additional relationships that the superclass does not have. For example, the subclasses, composite resource and individual resource, have additional relationships that the higher-level real resource and real resource element classes do not have. Furthermore, through these additional relationships, additional methods may be defined for the composite resource class and individual resource class that are not available to the real resource or real resource element classes.

Figure B.1 has a category drawn as a box in the upper left that is labeled virtual resource group. This category is an additional set of classes that shows the relationships to the virtual resource group. Figure B.2 shows the expansion of the virtual resource group. Each of the “dotted clouds” or classes was previously defined in Figure B.1, except for the virtual resource group.

The virtual resource group contains two or more virtual resources that must be defined simultaneously. Figure B.2 identifies the relationships between virtual resource group and some of the objects shown in Figure B.1. Figure B.2 does not identify the relationship among any other objects, since they remain the same as shown in Figure B.1. The expanded view and category are used to create a less complicated diagram.

B.2 Resource management object relationship models

The Booch OOA method [B1] uses object models to show how objects (instances) of the classes that are defined in the class model interact. In an object model, objects are shown as solid-line clouds, as opposed to the dotted-line clouds of the classes. Multiple instances of a class can be shown as “stacked” clouds. Lines between objects show a using relationship, with an arrow pointing to the object being used and text showing the messages being passed to the used object.

--`,,`-`-`,,`,,`,`,,`---
Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

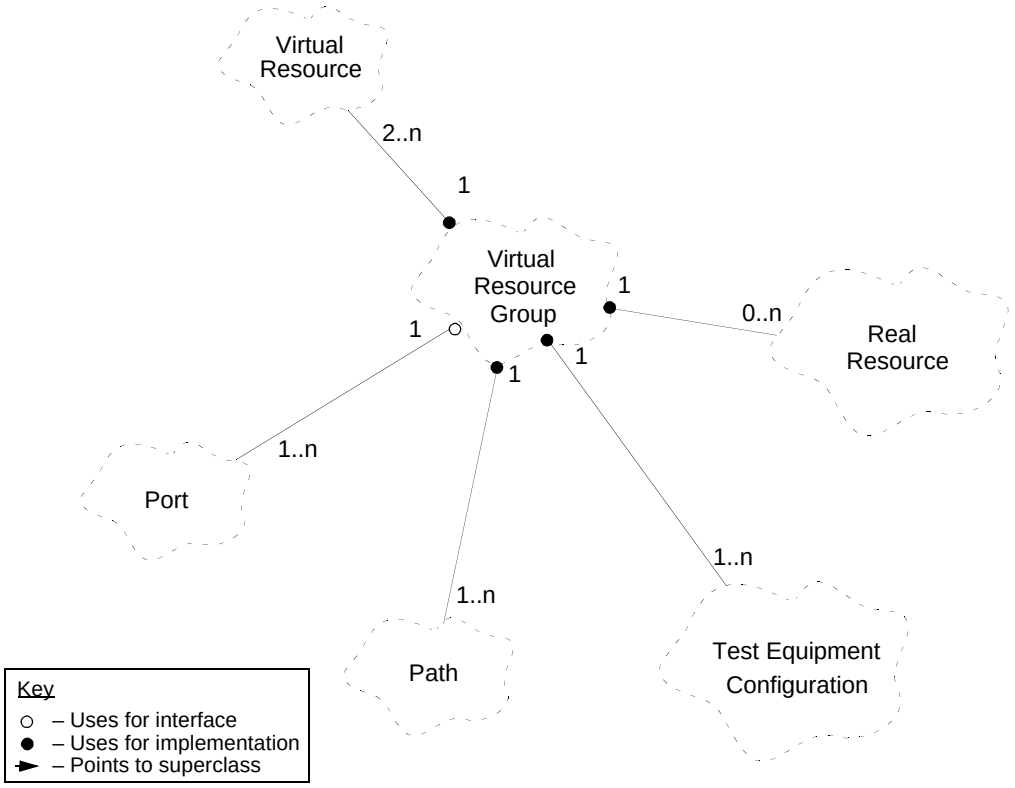

Figure B.2—Virtual resource group class model

Figures B.3, B.4, B.5, and B.6 show the following three different object relationships:

a) Virtual resource relation
b) Virtual resource group relation
c) Real resource relation

Each diagram identifies objects and their relationships from different views. For example, the virtual resource relation in Figure B.3 identifies how a test object would make use of a virtual resource object, and how that virtual resource becomes mapped to the physical resource object(s). Similarly, the real resource relation uses many of the same objects to show how a test object can directly access physical resource object(s) without the use of virtual resource objects. Each of these relationships is discussed in more detail in the following paragraphs.

Figure B.3 shows an object model for a test resource information object relation. The objects in this model relate directly to the classes shown in the class model (Figure B.1). The relationships shown in the diagram specify the messages passed between objects. These messages are the standard services that are described in a language-independent format within the specific ABBET standards. An example relationship, shown in Figure B.3, is between the test 1 object (of class test) and the virtual resource object (of class virtual resource). Test 1 uses the create method of the virtual resource class to create the virtual resource object and receives an identifier (VID) for that object. Test 1 can then use the initialize, destroy, or get_mapping virtual resource methods to manipulate the state of or retrieve information from the virtual resource object.

Additional relationships shown in Figure B.3 identify the virtual resource object call methods to get status, allocate, and deallocate the physical real resource objects and the physical path objects. To analyze which physical paths and resources to utilize, the virtual resource object requests the test equipment configuration object to find candidate resources and paths via the methods find_candidate_resources and

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

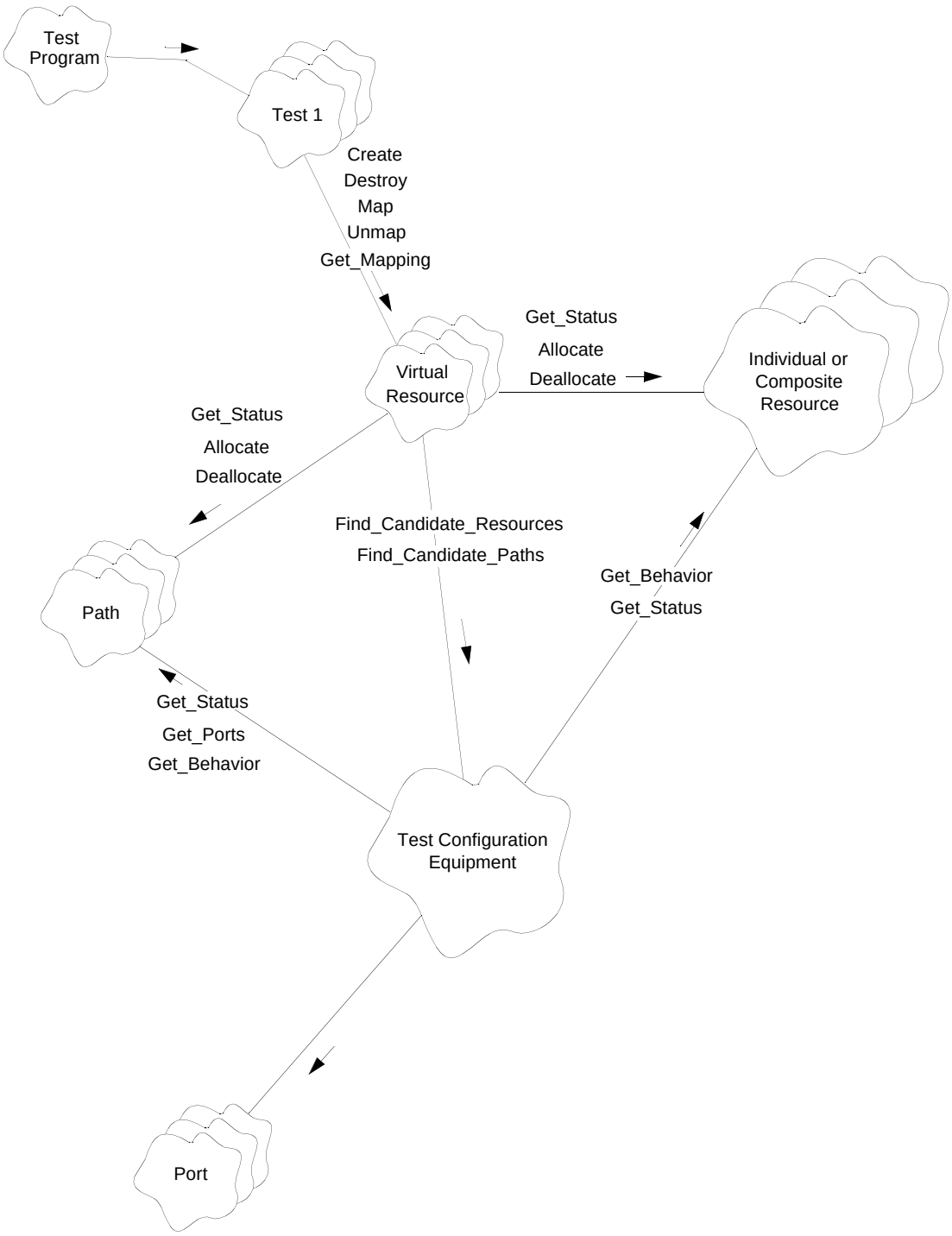

Figure B.3—Virtual resource object relation

find_candidate_paths. In order to identify potential physical paths and resources, the test equipment configuration object utilizes services to get the behavior and status of both the path object and the real resource object. For additional information, the test equipment configuration object uses the get_ports method associated with individual or composite resources object and the get_ports method associated with path object. From these methods, the test equipment configuration object is able to create a list of candidate paths and real resources that provide the behavior required by the virtual resource object.

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Figure B.4 shows the virtual resource group object relation. The objects and relationships are similar to those found in Figure B.3, which describe the virtual resource object relation. However, instead of the virtual resource object controlling the mapping to physical instruments, the virtual resource group object is in control. The test 1 object creates, destroys, adds virtual resources, maps, and unmaps the virtual resource group object. It is then the virtual resource group object that requests the test equipment configuration object to find candidate paths and real resources. In addition, it is the virtual resource group object that gets status from the real resource and path objects, and ultimately makes the decision on which paths and resources to allocate and deallocate.

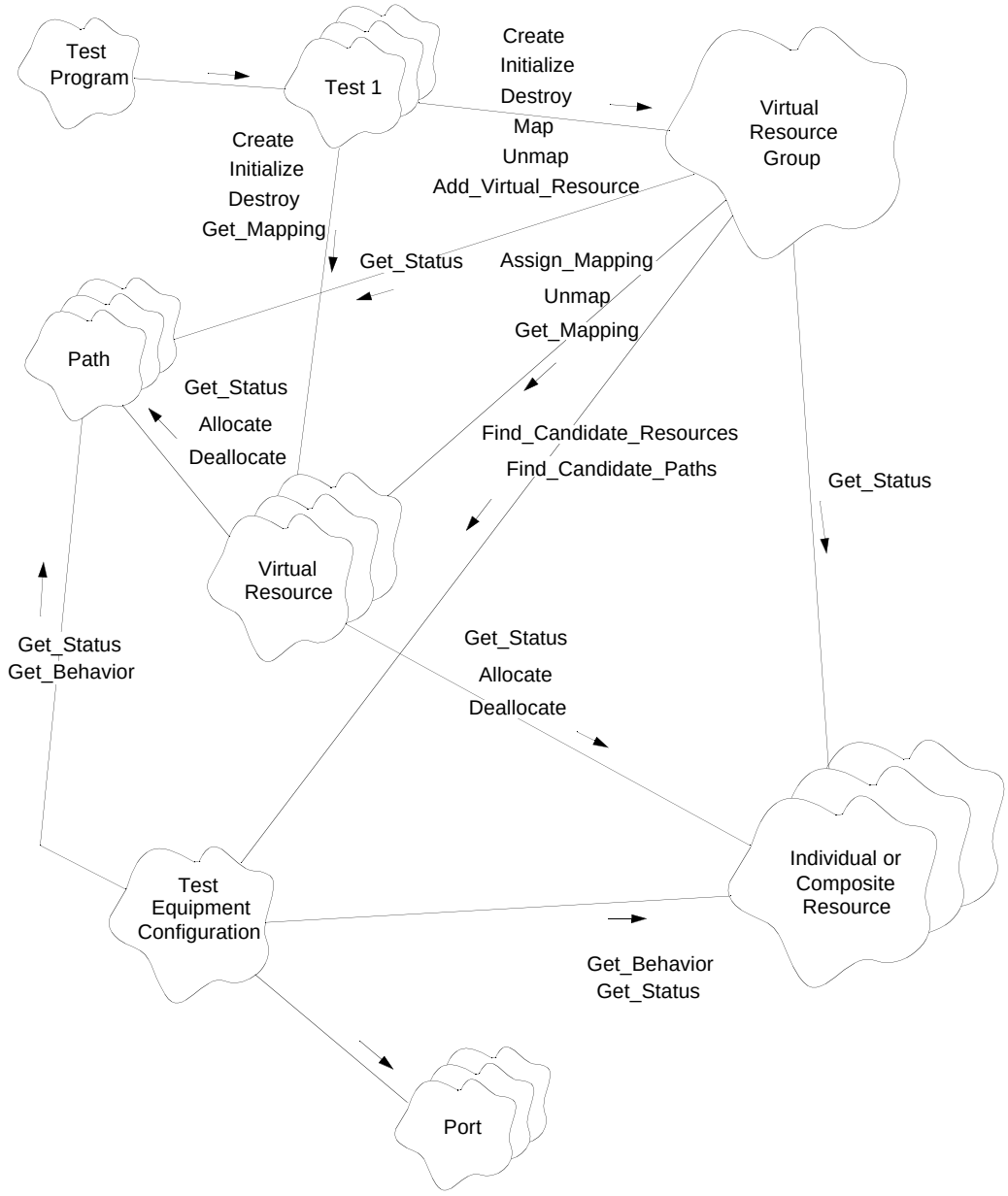

Figure B.4—Virtual resource group object relation

Figure B.5 shows the dynamic selection real resource object relation. This relation is depicted in Annex A, Clause A.5. As before, the objects are similar to the objects present in both Figures B.3 and B.4. All objects

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

that represent virtual entities, however, do not appear in Figure B.5. The test 1 object directly requests the test equipment configuration object to find candidate paths and resources. The test 1 object then gets status from the path object and the individual or composite resource object to make a decision on which physical instruments to use. Eventually, the test 1 object allocates and deallocates the path and individual or composite resource objects appropriately.

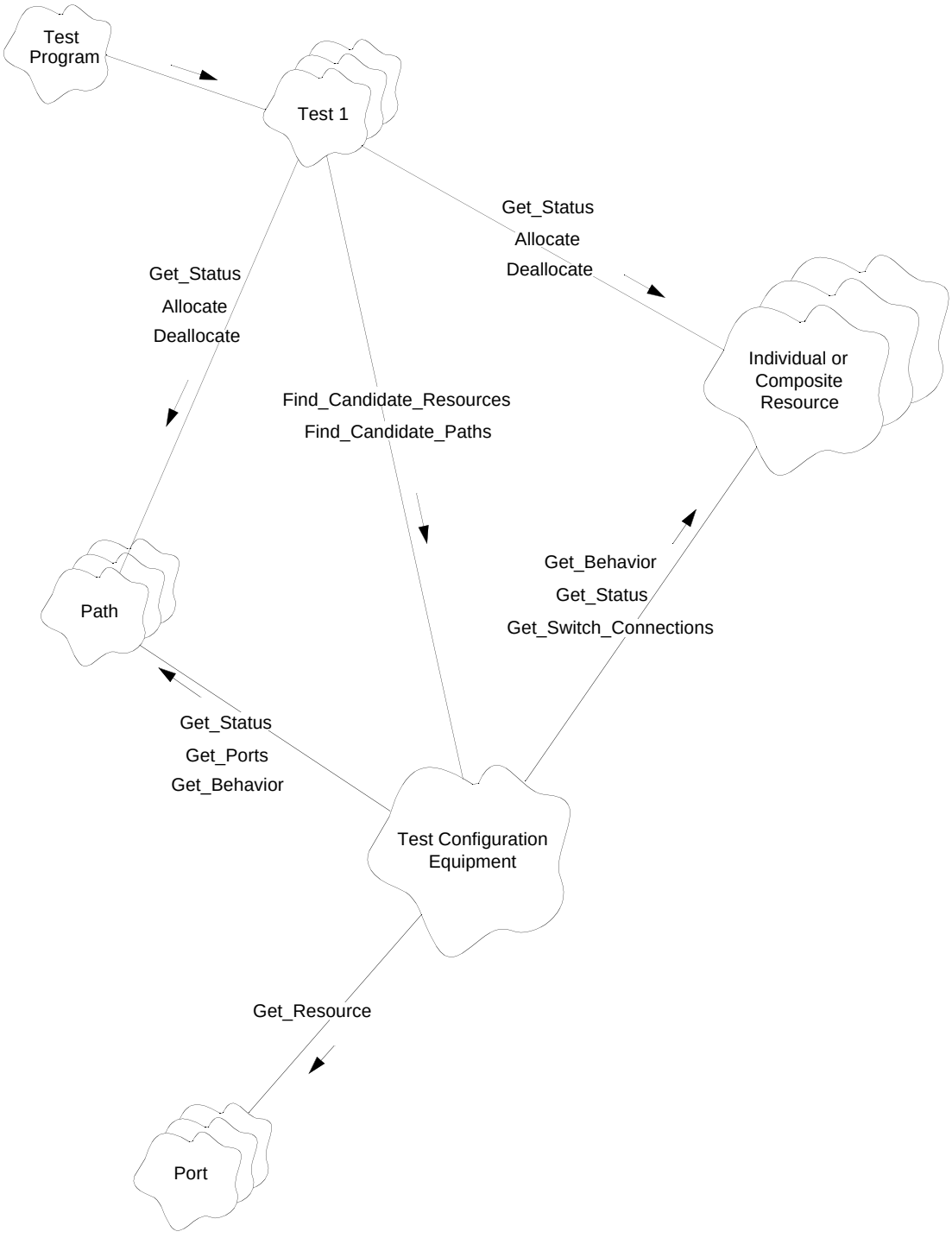

Figure B.5—Dynamic selection real resource object relation

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Figure B.6 shows the manual selection real resource object relation. This relation is depicted in Annex A, Clause A.6. The test program objects directly get status and appropriately allocate and deallocate the path and individual or composite resource objects. This requires test 1 to know the resource and path names within the test system. Programming at this level greatly reduces the portability of test 1.

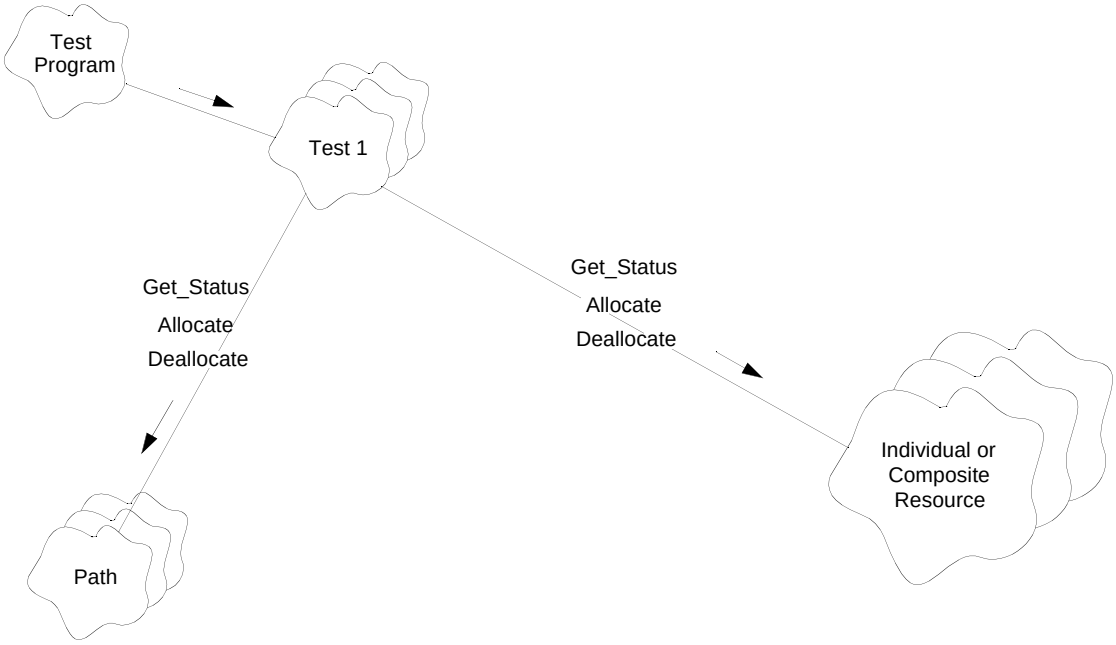

Figure B.6—Manual selection real resource object relation

The messages shown going between objects in Figures B.3, B.4, B.5, and B.6 are being documented as services in IEEE Std 1226.3-1998. The documentation of these services is the language-independent specifications found in Clause 6 of this standard.

--`,,`-`-`,,`,,`,`,,`---
Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Annex C

(normative)

Interface Definition Language (IDL) specification

This specification is a software interface for creating, accessing, manipulating, and destroying instances of the classes defined by the resource management software interface. It is described using IDL, as documented in OMG CORBA/IIOP Revision 2.1. The IDL representation is a language-neutral description of the interface, and shall be used as the basis for creating bindings to programming languages.

The specification includes IDL declarations that capture all the types, services, and inheritance relationships contained in the software interface. The types for “set” are represented using the IDL interface declarations that are included as part of the test foundation framework library described in Annex E.

The IDL specification does not formally capture every aspect of the resource management software interface. For example, the IDL syntax is not capable of representing optional attributes, uniqueness constraints, or rules. Implementations of this interface should ensure that instances of the software interface do not violate these conditions whenever practical. The software interface contains textual descriptions that apply to IDL.

The specification consists of multiple IDL files with definitions of modules and interfaces. A module contains one or more interfaces and is used for scoping. An interface is analogous to a class. Not all interfaces are defined in modules. This is to allow forward referencing. If there is no module defined in the file, then the file name, in lower-case letters, is used for referencing. Inter-module references exist for the purposes of representing inheritance and object references. The modules and the inter-module references are shown in Figure C.1.

The compilation order is:

Module name	**File name**	**comments**
rmsub	rmsub.idl	definitions only
RM_Port	rmport.idl	
RM_C	rmcnx.idl	
RM_Path	rmpath.idl	
RM_RR	rmrelrs.idl	
RM_VR	rmvrtrs.idl	
RM_VRG	rmvrtgp.idl	
RM_TEC	rmtstcf.idl	

Table C.1 indicates the allowed subsets and shows which interfaces are required to be supported for each allowed subset. The numbers across the top of Table C.1 are the allowed subsets and combinations of subsets defined in Clause 5. The “X” indicates that the software package is required for implementation of the subset.

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

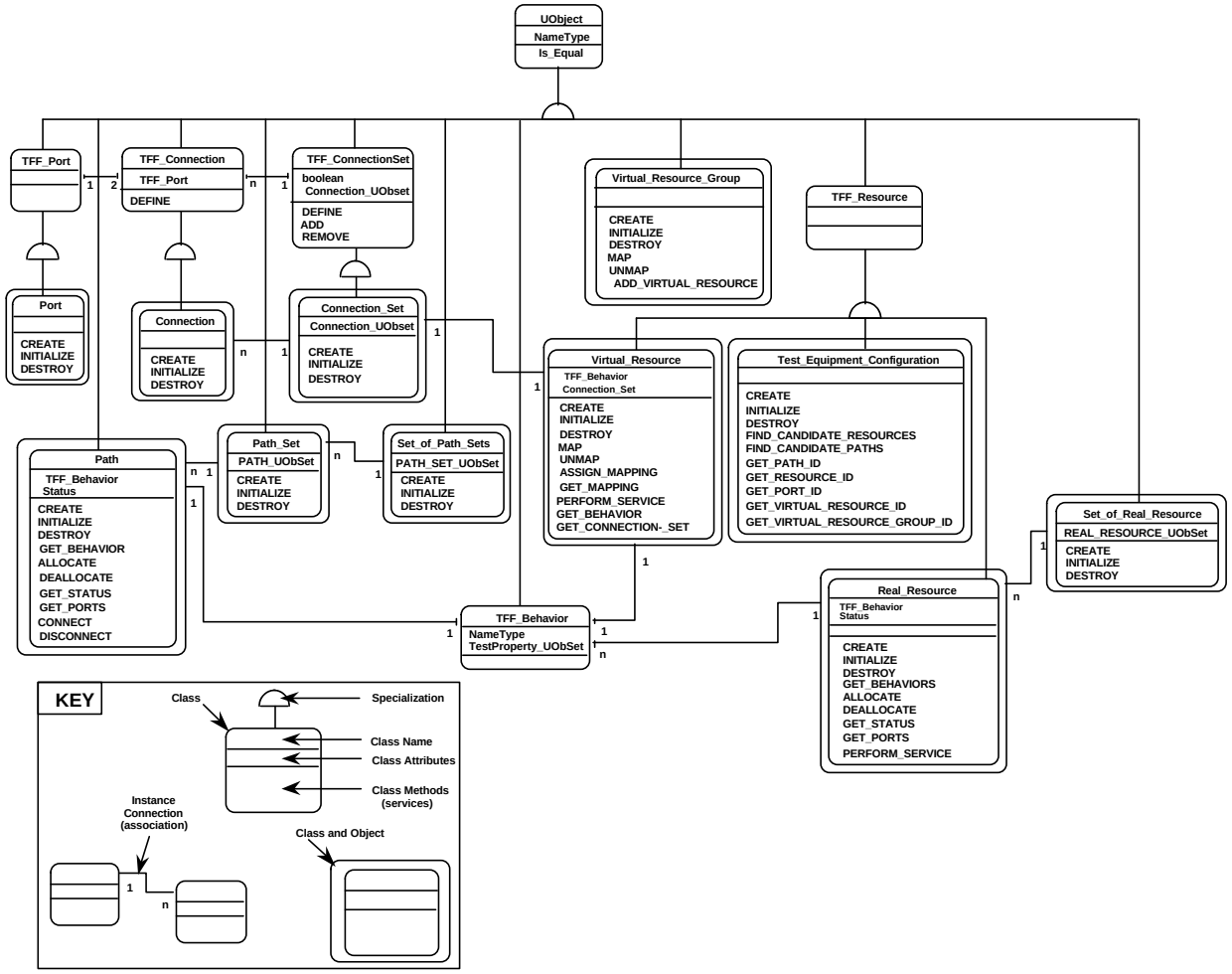

Figure C.1—Inter-module references

Table C.1—IDL conformance table

		1	2	3	4	5	6	7
I	Port	X	X	X	X	X	X	X
N	Connection	X	X	X		X	X	X
T	Connection set	X	X	X		X	X	X
E	Path			X	X	X	X	X
R	Path set			X	X	X	X	X
F	Set of paths			X	X	X	X	X
A	Real resource			X	X	X	X	X
C	Set of real resources			X	X	X	X	X
E	Virtual resource	X	X			X	X	X
S	Virtual resource group		X					X
	Test equipment configuration			X		X		

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

C.1 Definitions

C.1.1 Virtual resource management services

C.1.1.1 [RM_VR] module

The RM_VR module contains the virtual resource class (see Table C.2). The services assign_mapping and get_mapping are conditionally defined based on the subsets being implemented. The get_virtual_resource_id service is defined in RM_TEC module. Table C.3 contains the IDL/Clause 6 mapping for virtual resources.

Table C.2—RM_VR module represented in IDL

```
#include "rmsub.idl"
#if defined(_RM_SUBSET1) || defined(_RM_SUBSET2)
#ifndef _RM_VRTRSC_IDL
#define _RM_VRTRSC_IDL
#include "rmcnx.idl"
#include "stdtypes.idl"
#include "tffbehav.idl"
#include "tffres.idl"
#if defined(_RM_SUBSET3) || defined(_RM_SUBSET4)
 #include "rmpath.idl"
 #include "rmrelrs.idl"
#endif
//
module RM_VR
{
   interface Virtual_Resource;
      exception Invalid_State {};
      exception Invalid_Virtual_Resource {}; /* ie resource not created yet */
   //
   //-------------------------------------------------------------------
   // Virtual_Resource Class
   //-------------------------------------------------------------------
   //
   interface Virtual_Resource : TFF_Resource::Resource
   {
      readonly attribute TFF_Behavior::Behavior          Behavior;
      readonly attribute RM_C::Connection_Set            Connection_Set_ID;
      exception Invalid_Behavior {};
      exception Invalid_Parameters {};
      exception Invalid_Name {};             /* name is too big or incorrect */
      exception Invalid_Service_ID {};
      exception Unsuccessful_Operation {};   /*ie unable to create,destroy,etc*/

      void        Initialize
         ( in     NameType                Name,
           in     TFF_Behavior::Behavior  With_Behavior,
           in     RM_C::Connection_Set    Set_of_Connections )
            raises (Invalid_Behavior,
                    RM_C::Invalid_Connection_Set,
                    Invalid_Name,
                    Invalid_State,
                    Unsuccessful_Operation);
      /* create/destroy provided by constructor/destructor */

void        Map()
            raises (Invalid_State, Invalid_Virtual_Resource, Unsuccessful_Operation);

      void        Unmap()
            raises (Invalid_State, Invalid_Virtual_Resource, Unsuccessful_Operation);
```

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Table C.2—RM_VR module represented in IDL *(continued)*

```
      void        Perform_Service
         ( in     Services   Service_to_Perform,
           in     AnyList            Input_Parameters,
           out    AnyList            Data )
            raises (Invalid_Parameters,
                    Invalid_Service_ID,
                    Invalid_State,
                    Invalid_Virtual_Resource,
                    Unsuccessful_Operation);

#if defined(_RM_SUBSET3) || defined(_RM_SUBSET4)
      void        Assign_Mapping
         ( in     RM_RR::Real_Resource Resource_Mapping,
           in     RM_Path::Set_of_Path_Sets
                                       Connection_Mapping )

raises (RM_Path::Invalid_Set_of_Path_Sets,
                    Invalid_State,
                    RM_RR::Invalid_Real_Resource,
                    Invalid_Virtual_Resource,
                    Unsuccessful_Operation );

      void        Get_Mapping
         ( out    RM_RR::Real_Resource Resource_Mapping,
           out    RM_Path::Set_of_Path_Sets
                                       Connection_Mapping )
            raises (Invalid_State,
                    Invalid_Virtual_Resource,
                    Unsuccessful_Operation );

#endif   // defined(_RM_SUBSET3) || defined(_RM_SUBSET4)
   };
};
#endif   // ifndef _RM_VRTRSC_IDL
#endif   // defined(_RM_SUBSET1) || defined(_RM_SUBSET2)
```

Table C.3—IDL/Clause 6 mapping for virtual resource services

IDL operation	Service
Constructor	6.2.1.2
Destructor	6.2.1.4
Initialize	6.2.1.3
Map	6.2.1.5
Unmap	6.2.1.6
Perform_service	6.2.1.9
Assign_mapping	6.2.1.7
Get_mapping	6.2.1.8

C.1.1.2 [RM_C] module

The RM_C module contains the connection and connection_set classes (see Table C.4). Table C.5 contains the IDL/Clause 6 mapping for connection services.

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Table C.4—RM_C module represented in IDL

```
#ifndef _RM_CONNECT_IDL
#define _RM_CONNECT_IDL
#include "collect.idl"
#include "rmport.idl"
#include "stdtypes.idl"
#include "tffconn.idl"
//
module RM_C
{
   interface Connection;
   interface Connection_Set;
   exception Invalid_Name {};
   exception Invalid_Connection {};     /* Connection invalid */
   exception Invalid_Connection_Set {};
   exception Invalid_State {};
   //
   //---------------------------------------------------------------
   // Connection Class
   //---------------------------------------------------------------
   //
   interface Connection          : TFF_Connection::Connection
   {
      exception Unsuccessful_Operation {};/* ie unable to create,destroy,etc */

      void        Initialize
         ( in     NameType               Name,
           in     RM_Port::Port          From_Port,
           in     RM_Port::Port          To_Port )
            raises (Invalid_Connection,
                    Invalid_Name,
                    RM_Port::Invalid_Port,
                    Invalid_State,
                    Unsuccessful_Operation);

      /* create/destroy provided by constructor/destructor */
   };
//
   //---------------------------------------------------------------
   // Connection_Set Class
   //---------------------------------------------------------------
   //
   interface Connection_Set          : TFF_Connection::Connection_Set
   {
};
};
#endif;
```

Table C.5—IDL/Clause 6 mapping for connection services

IDL operation	Service
Constructor connection	6.2.2.2
Destructor	6.2.2.4
Initialize	6.2.2.3
Constructor connection_set	6.2.3.2
Destructor	6.2.3.4
Initialize	6.2.3.3

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

C.1.1.3 Port interfaces

This file contains the port class and set of ports (see Table C.6). Table C.7 contains the IDL/Clause 6 mapping for port services.

Table C.6—[RM Port] module represented in IDL

```
#ifndef _RM_PORT_IDL
#define _RM_PORT_IDL
#include "collect.idl"
#include "stdtypes.idl"
#include "tffport.idl"
//
module RM_Port
{
      exception Invalid_Port {};          /* port has not been created */
      exception Invalid_State {};
   //
   //-------------------------------------------------------------------
   // Port Class
   //-------------------------------------------------------------------
   //
   interface Port               : TFF_Port::Port
   {
      exception Invalid_Name {};          /* name is too big or incorrect */
      exception Unsuccessful_Operation {};/* ie unable to create,etc */

      void        Initialize
         ( in     NameType               Name )
            raises (Invalid_Name,
                    Invalid_Port,
                    Invalid_State,
                    Unsuccessful_Operation);

      /* create/destroy provided by constructor/destructor */

   };
   //
   //-------------------------------------------------------------------
   // Set of Ports
   //-------------------------------------------------------------------
   //
   DEFINETYPEDSET(Port)

};
#endif
```

Table C.7—IDL/Clause 6 mapping for port services

IDL operation	Service
Constructor	6.2.4.2
Destructor	6.2.4.4
Initialize	6.2.4.3

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

C.1.1.4 [RM_VRG] module

The RM_VRG module contains the virtual resource group class (see Table C.8). Table C.9 contains the IDL/Clause 6 mapping for virtual resource group services.

Table C.8—RM_VRG module represented in IDL

```
#include "rmsub.idl"
#ifdef _RM_SUBSET2
#ifndef _RM_VRTGRP_IDL
#define _RM_VRTGRP_IDL
#include "rmcnx.idl"
#include "rmvrtrs.idl"
#include "stdtypes.idl"
//
module RM_VRG
{
   interface Virtual_Resource_Group;
      exception Invalid_Name {};            /* name is too big or incorrect */
      exception Invalid_State {};
   //
   //-------------------------------------------------------------------
   // Virtual_Resource_Group Class
   //-------------------------------------------------------------------
   //
   interface Virtual_Resource_Group : UObject
   {
      exception Unsuccessful_Operation {};   /*ie unable to create,destroy,etc*/
      exception Invalid_Virtual_Res_Grp {};  /* ie resource not created yet */

      void        Initialize
         ( in     NameType            Name )
            raises (Invalid_Name,
                    Invalid_State,
                    Invalid_Virtual_Res_Grp,
                    Unsuccessful_Operation);

      /* create/destroy provided by constructor/destructor */

      void        Map()
            raises (Invalid_Virtual_Res_Grp, Invalid_State, Unsuccessful_Operation);

      void        Unmap()
            raises (Invalid_Virtual_Res_Grp, Invalid_State, Unsuccessful_Operation);

      void        Add_Virtual_Resource
         ( in     RM_VR::Virtual_Resource    Virtual_Resource,
           in     RM_C::Connection_Set       Connection_Set )
            raises (Invalid_Virtual_Res_Grp,
                    RM_C::Invalid_Connection_Set,
                    Invalid_State,
                    RM_VR::Invalid_Virtual_Resource,
                    Unsuccessful_Operation);

 void        Remove_Virtual_Resource
         ( in     RM_VR::Virtual_Resource    Virtual_Resource )
            raises (Invalid_State,
                    Invalid_Virtual_Res_Grp,
                    RM_VR::Invalid_Virtual_Resource,
                    Unsuccessful_Operation);

   };
};
#endif   // ifndef _RM_VRTGRP_IDL
#endif   // ifdef _RM_SUBSET2
```

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Table C.9—IDL/Clause 6 mapping for virtual resource group

IDL operation	Service
Constructor	6.2.5.2
Destructor	6.2.5.4
Initialize	6.2.5.3
Map	6.2.5.5
Unmap	6.2.5.6
Add_virtual_resource	6.2.5.7
Remove_virtual_resource	6.2.5.8

C.1.2 Test equipment configuration management services

C.1.2.1 [RM_TEC] module

The RM_TEC module contains the test equipment configuration class (see Table C.10). Table C.11 contains the IDL/Clause 6 mapping for test equipment configuration services.

Table C.10—RM_TEC module represented in IDL

```
#include "rmsub.idl"
#ifdef _RM_SUBSET3
#ifndef _RM_TESTCFG_IDL
#define _RM_TESTCFG_IDL
#include "rmcnx.idl"
#include "rmpath.idl"
#include "rmport.idl"
#include "rmrelrs.idl"
#include "stdtypes.idl"
#include "tffbehav.idl"
#include "tffres.idl"
// if subset1 and subset3 (subset5)
#ifdef _RM_SUBSET1
 #include "rmvrtrs.idl"
#endif
// if subset2 and subset3 (all, no subset)
#ifdef _RM_SUBSET2
 #include "rmvrtgp.idl"
#endif
//
module RM_TEC
{
   interface Test_Equipment_Configuration;
   //
   //-------------------------------------------------------------------
   // Test_Equipment_Configuration Class
   //-------------------------------------------------------------------
   //
   interface Test_Equipment_Configuration : TFF_Resource::Resource
   {
      enum Allocation_Condition { All_Resources, Non_Allocated };

      exception Invalid_Behavior {};
      exception Invalid_Name {};             /* name is too big or incorrect */
      exception Invalid_State {};
      exception Invalid_Test_Config {};      /* test equip config not created */
      exception Unsuccessful_Operation {};   /*ie unable to create,destroy,etc*/
```


Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Table C.10—RM_TEC module represented in IDL *(continued)*

```
      void        Initialize
         ( in     NameType            Name )
            raises (Invalid_Name,
                    Invalid_State,
                    Invalid_Test_Config,
                    Unsuccessful_Operation);

      /* create/destroy provided by constructor/destructor */

      RM_RR::Set_of_Real_Resources   Find_Candidate_Resources
         ( in    TFF_Behavior::Behavior         With_Behavior,
           in    Allocation_Condition           Search_Criteria )
            raises (Invalid_Behavior,
                    Invalid_State,
  Invalid_Test_Config,
                    Unsuccessful_Operation);

      RM_Path::Set_of_Path_Sets  Find_Candidate_Paths
         ( in    RM_RR::Real_Resource  Resource,
           in    RM_C::Connection_Set  Required_Connections )
            raises (RM_C::Invalid_Connection_Set,
                    RM_RR::Invalid_Real_Resource,
                    Invalid_State,
                    Invalid_Test_Config,
                    Unsuccessful_Operation);

      RM_Path::Path Get_Path_ID
         ( in    RM_C::Connection      Required_Endpoints )
            raises (RM_C::Invalid_Connection,
                    Invalid_State,
                    Invalid_Test_Config,
                    Unsuccessful_Operation);

      RM_RR::Real_Resource  Get_Resource_ID
         ( in    NameType              Name )
            raises (RM_RR::Invalid_Name,
                    Invalid_State,
                    Invalid_Test_Config,
                    Unsuccessful_Operation);

      RM_Port::Port    Get_Port_ID
         ( in    NameType              Name )
            raises (Invalid_Name,
                    Invalid_State,
                    Invalid_Test_Config,
                    Unsuccessful_Operation);
#ifdef _RM_SUBSET1
      RM_VR::Virtual_Resource  Get_Virtual_Resource_ID
         ( in     NameType             Name )
            raises (RM_VR::Invalid_Name,
                    Invalid_State,
                    Unsuccessful_Operation);
#endif
#ifdef _RM_SUBSET2
      RM_VRG::Virtual_Resource_Group
                  Get_Virtual_Resource_Group_ID
         ( in     NameType             Name )
            raises (RM_VRG::Invalid_Name,
                    Invalid_State,
                    Unsuccessful_Operation);
#endif
   };
};
#endif   // ifndef _RM_TESTCFG_IDL
#endif   // ifdef _RM_SUBSET3
```

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

Table C.11—IDL/Clause 6 mapping for test equipment configuration services

IDL operation	Service
Constructor	6.3.1.2
Destructor	6.3.1.4
Initialize	6.3.1.3
Find_candidate_resources	6.3.1.5
Find_candidate_paths	6.3.1.6
Get_path_id	6.3.1.7
Get_resource_id	6.3.1.8
Get_port_id	6.3.1.9
Get_virtual_resource_id	6.3.1.11
Get_virtual_resource_group_id	6.3.1.10

C.1.3 Real resource management services

C.1.3.1 [RM_Path] module

The RM_Path module contains the path, path_set, and set_of_path_sets classes (see Table C.12). Table C.13 contains the IDL/Clause 6 mapping for path services.

Table C.12—RM_Path module represented in IDL

```
#include "rmsub.idl"
#if defined(_RM_SUBSET3) || defined(_RM_SUBSET4)
#ifndef _RM_PATH_IDL
#define _RM_PATH_IDL
#include "collect.idl"
#include "rmport.idl"
#include "stdtypes.idl"
#include "tffbehav.idl"
//
module RM_Path
{
   interface Path;
   interface Path_Set;
   interface Set_of_Path_Sets;
      exception Invalid_Name {};          /* name is too big or incorrect */
      exception Invalid_Path {};             /* path has not been created */
      exception Invalid_Path_Set {};         /* path set has not been created */
      exception Invalid_Set_of_Path_Sets {};
      exception Invalid_State {};
   //
   //--------------------------------------------------------------------
   // Path Class
   //--------------------------------------------------------------------
   //
```


Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Table C.12—RM_Path module represented in IDL *(continued)*

```
   interface Path : UObject
   {
      enum Status { Available, Unavailable };

      readonly attribute TFF_Behavior::Behavior Behavior;
      readonly attribute Status                 Path_Status;

      exception Unsuccessful_Operation {};   /* ie unable to create,etc */

      void        Initialize
         ( in     NameType               Name )
            raises (Invalid_Name,
                    Invalid_State,
                    Unsuccessful_Operation);

      /* create/destroy provided by constructor/destructor */
      /* get_behavior provided for readonly attribute */

     void        Allocate()
            raises (Invalid_Path, Invalid_State, Unsuccessful_Operation);

      void        Deallocate()
            raises (Invalid_Path, Invalid_State, Unsuccessful_Operation);

      /* get_status provided for readonly attribute */

      void        Get_Ports
         ( out    RM_Port::Port     Port_1,
           out    RM_Port::Port     Port_2 )
            raises (Invalid_Path, Invalid_State, Unsuccessful_Operation);

      void        Connect()
            raises (Invalid_Path, Invalid_State, Unsuccessful_Operation);

      void        Disconnect()
            raises (Invalid_Path, Invalid_State, Unsuccessful_Operation);

};
   //
   //-------------------------------------------------------------------
   // Path_Set Class
   //-------------------------------------------------------------------
   //
   DEFINETYPEDSET( Path )        /* Path Set */
   interface Path_Set            : UObject
   {
      attribute Path_UObSet      Paths;
      exception Unsuccessful_Operation {};/* ie unable to create,etc */

      void        Initialize
         ( in     NameType               Name,
           in     Path_UObSet            Paths )
            raises (Invalid_Name,
                    Invalid_Path,
                    Invalid_Path_Set,
                    Invalid_State,
                    Unsuccessful_Operation);

      /* create/destroy provided by constructor/destructor */
   };
   //
   //-------------------------------------------------------------------
   // Set_Path_Set Class
   //-------------------------------------------------------------------
   //
```

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

Table C.12—RM_Path module represented in IDL *(continued)*

```
   DEFINETYPEDSET( Path_Set )    /* Set of Path Sets */
   interface Set_of_Path_Sets    : UObject
   {
      attribute Path_Set_UObSet      Path_Sets;
      exception Unsuccessful_Operation {};/* ie unable to create,etc */

      void        Initialize
         ( in     NameType               Name,
           in     Path_Set_UObSet        Set_of_Paths )
            raises (Invalid_Name,
                    Invalid_Path_Set,
                    Invalid_Set_of_Path_Sets,
                    Invalid_State,
                    Unsuccessful_Operation);

      /* create/destroy provided by constructor/destructor */
   };
};
#endif   // ifndef _RM_PATH_IDL
#endif   // defined(_RM_SUBSET3) || defined(_RM_SUBSET4)
```

Table C.13—IDL/Clause 6 mapping for path services

IDL Operation	Service
Constructor path	6.4.1.2
Destructor	6.4.1.4
Initialize	6.4.1.3
Get_behavior	6.4.1.5
Allocate	6.4.1.6
Deallocate	6.4.1.7
Get_status	6.4.1.8
Get_ports	6.4.1.9
Connect	6.4.1.10
Disconnect	6.4.1.11
Constructor path set	6.4.2.2
Destructor	6.4.2.4
Initialize	6.4.2.3
Constructor set of path sets	6.4.3.2
Destructor	6.4.3.4
Initialize	6.4.3.3

--`,,`-`-`,,`,,`,`,,`---

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

C.1.3.2 [RM_RR] module

The RM_RR module contains the real resource and set of real resources classes (see Table C.14). Table C.15 contains the IDL/Clause 6 mapping for real resource services.

Table C.14—RM_RR module represented in IDL

```
#include "rmsub.idl"
#if defined(_RM_SUBSET3) || defined(_RM_SUBSET4)
#ifndef _RM_REALRSC_IDL
#define _RM_REALRSC_IDL
#include "collect.idl"
#include "rmport.idl"
#include "stdtypes.idl"
#include "tffbehav.idl"
#include "tffres.idl"
//
module RM_RR
{
   interface Real_Resource;
   interface Set_of_Real_Resources;
      exception Invalid_Name {};             /* name is too big or incorrect */
      exception Invalid_Real_Resource {};    /* not created */
      exception Invalid_Real_Resource_Set {};
      exception Invalid_State {};
   //
   //-----------------------------------------------------------------
   // Real_Resource Class
   //-----------------------------------------------------------------
   //
   interface Real_Resource       : TFF_Resource::Resource
   {
      enum Status { Available, Unavailable };
      readonly attribute TFF_Behavior::Behavior_UObSet  Behaviors;
      readonly attribute Status                         Real_Resource_Status;
      exception Invalid_Behavior {};
      exception Invalid_Parameters {};
      exception Invalid_Service_ID {};
      exception Unsuccessful_Operation {};   /* ie unable to create,etc */

      void        Initialize
         ( in     NameType             Name )
            raises (Invalid_Name,
                    Invalid_Real_Resource,
                    Invalid_State,
                    Unsuccessful_Operation);

      /* create/destroy provided by constructor/destructor */
      /* get_behaviors provided by readonly attribute */

      void        Allocate()
            raises (Invalid_Real_Resource, Invalid_State, Unsuccessful_Operation);


      void        Deallocate()
            raises (Invalid_Real_Resource, Invalid_State, Unsuccessful_Operation);

/* get_status provided for readonly attribute */

      RM_Port::Port_UObSet Get_Ports
         ( in     RM_Port::Port          Of_Port )
            raises (Invalid_Real_Resource,
                    RM_Port::Invalid_Port,
                    Invalid_State,
                    Unsuccessful_Operation);
```

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Table C.14—RM_RR module represented in IDL *(continued)*

```
      void        Perform_Service
         ( in     Services   Service_to_Perform,
   in     TFF_Behavior::Behavior   With_Behavior,
          in     RM_Port::Port_UObSet     At_Ports,
          in     AnyList            Input_Parameters,
          out    AnyList            Data )
           raises (Invalid_Behavior,
                   Invalid_Parameters,
                   RM_Port::Invalid_Port,
                   Invalid_Real_Resource,
                   Invalid_Service_ID,
                   Invalid_State,
                   Unsuccessful_Operation);

   };
   //
   //-----------------------------------------------------------------
   // Real_Resource_Set Class
   //-----------------------------------------------------------------
   //
   DEFINETYPEDSET(Real_Resource)
   interface Set_of_Real_Resources  : UObject
   {
      attribute Real_Resource_UObSet   Real_Resource;
      exception Unsuccessful_Operation {};/* ie unable to create,etc */

      void        Initialize
         ( in     NameType             Name,
           in     Real_Resource_UObSet    Real_Resources )
           raises (Invalid_Name,
                   Invalid_Real_Resource,
                   Invalid_Real_Resource_Set,
                   Invalid_State,
                   Unsuccessful_Operation);

      /* create/destroy provided by constructor/destructor */
   };
};
#endif   // ifndef _RM_REALRSC_IDL
#endif   // defined(_RM_SUBSET3) || defined(_RM_SUBSET4)
```


Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Table C.15—IDL/Clause 6 mapping for real resources

IDL operation	Service
Constructor real_resource	6.4.4.2
Destructor	6.4.4.4
Initialize	6.4.4.3
Get_behaviors	6.4.4.5
Allocate	6.4.4.6
Deallocate	6.4.4.7
Get_status	6.4.4.8
Get_ports	6.4.4.9
Perform_service	6.4.4.10
Constructor set of real resource	6.4.5.2
Destructor	6.4.5.4
Initialize	6.4.5.3

C.1.4 Support services

C.1.4.1 RM Subset definitions

The RM subset definitions are used to control the conditional compilation of the IDL files, based on the subsets being implemented. Table C.16 contains the RM subsets represented in IDL. If subset 1, virtual resource, is being implemented, then the #define for subset 1 would be provided, and the other subset definitions would be commented out. The default setting is for subsets 2 and 3 defined, which means that all of the services are being provided.

Table C.16—RM subsets represented in IDL

```
//    Subset 1 = Virtual Resource
//#define _RM_SUBSET1
//
//    Subset 2 = Virtual Resource, Virtual Resource Group
#define _RM_SUBSET2
//
//    Subset 3 = Real Resource Mgmt, Test Equipment Configuration
#define _RM_SUBSET3
//
//    Subset 4 = Real Resource Mgmt
//#define _RM_SUBSET4
```

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

Annex D

(informative)

Bibliography

[B1] Booch, Grady, *Object Oriented Design With Applications.* Redwood City, CA: Benjamin/Cummings, 1991.

Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Annex E

(informative)

Test foundation framework

This annex contains the IDL associated with the test foundation framework as defined in IEEE P1226/D 15.3.

The compilation order is:

Module name	File name
Stdtypes	stdtypes.idl
Collection	collect.idl
TFF_Port	tffport.idl
TFF_Connection	tffconn.idl
TFF_Resource	tffres.idl
TFF_TestProperty	tfftprop.idl
TFF_Behavior	tffbehav.idl
TFF_Signal	tffsignl.idl

E.1 Stdtypes

Table E.1 contains the IDL for stdtypes.

Table E.1—Stdtypes module represented in IDL

```
#ifndef _STDTYPES_H
#define _STDTYPES_H

typedef string NameType;               // Type for names
typedef sequence<any> AnyList;
const string NoName = "NONAME";  // Default name for things that
                                 // have no name assigned
                                 // have no name assigned.
const long One = 1;                                    // Constant value 1
const long Zero = 0;                                   // Constant value 0

enum InInterval { LOW,MID,HIGH };
       // Interval test return values, Interval is defined by [low..high]
       // LOW:  value < low
       // IN:   low <= value <= high
       // HIGH: high < value

// Implementation_Defined
// This class is a root for the entire hierarchy, and is provided here
// only as a forward reference.  The implementor is free to define this
// as needed to provide a base for the UObject derived class shown
// below.  For example, an implementation based on a commercially available
// ORB would substitute the name of the root class defined by that ORB
// in the place of the symbol Implementation_Defined.  An implementation
// not based on an ORB could comment out the forward reference and
// remove the inheritance syntax on the UObject definition.
```

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Table E.1—Stdtypes module represented in IDL *(continued)*

```
interface Implementation_Defined
  {};

// UObject
// The mother of all objects used in the TFCL

interface UObject : Implementation_Defined
{
   attribute NameType mName;
   boolean IsEqual(in UObject anObject);    // Answer true if object == anObject
};

enum Engineering_Units
{
   METER,
   SEC
};
enum Services
{
   SETUP,
   CHANGE,
   ARM,
   FETCH,
   RESET,
   APPLY,
   MEASURE,
   MONITOR,
   READ,
   INITIATE,
   ENABLE,
   DISABLE
};

#endif  // _STDTYPES_H
```

E.2 Collection

Table E.2 contains the IDL for collection.

Table E.2—Collection module represented in IDL

```
#ifndef _COLLECT_IDL
#define _COLLECT_IDL

#include "stdtypes.idl"

module Collection {

    interface UObArray;
    interface UObList;
    interface UObSet;
    interface UObBag;

    // Object array
    // All elements stored in an object array are derived from UObject, as is
    // the array itself.

    interface UObArray : UObject
    {
     attribute long mSize;          // Number of elements in the array.
     attribute long mUpperBound;    // Maximum number of elements in the array
```


Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Table E.2—Collection module represented in IDL *(continued)*

```
    UObject GetAt(in long nIndex);         // Answer the element at position nIndex
    void SetAt(in long nIndex,in UObject anElement); // Set the value of element at
                                             // nIndex to value anElement
    long Add(in UObject anElement);        // Add a new element to the array
    boolean IsEmpty(in long nIndex);       // Answers true if array location is empty
    void InsertAt(in long nIndex,in UObject anElement); // Insert anElement at nIndex
      void RemoveAt(in long nIndex,in long aCount);   // Remove aCount elements starting at
                                          // nIndex
   };

   // UObList
   // A list of things.

   interface UObList : UObject
   {
      attribute long mSize;                // Number of elements in the list
      attribute long mUpperBound;          // Maximum number of elements in the list
long AddHead(in UObject anElement);     // Add an element at the head of the list
       long AddTail(in UObject anElement);    // Add an element at the tail of the list
       long Find(in UObject searchValue,in long startAfter);   // Find the position of
                                             // of a specific object in the list

UObject GetAt(in long position);        // Answer the element at position
       UObject GetHead();               // Answer the element at the head of the list
       UObject GetNext(inout long position);  // Get element, set position to next
       UObject GetPrev(inout long position);  // Get element, set position to previous
       UObject GetTail();                     // Get element at the tail of the list
       long InsertAfter(in long position,in UObject anElement);   // Insert an element
                                             // after the specified position
       long InsertBefore(in long position,in UObject anElement);  // Insert an element
                                            // before the specified position.
     void RemoveAll();                      // Remove all elements from the list
     void RemoveAt(in long position);       // Remove the element at position
     void RemoveHead();                     // Remove the element at the head
     void SetAt(in long position,in UObject anElement); // Set the value of an element
                                             // at position
   };

   // UObBag
   // A UObBag is a collection which allows duplicate members.

   interface UObBag : UObject
   {
      attribute long mSize;                 // Number of elements in the set
      attribute long mUpperBound;           // Maximum number of elements in the set

      void Add(in UObject anElement);      // Add anElement to the set
      void Remove(in UObject anElement);   // Remove anElement from the set
  boolean IsMember(in UObject anElement);   // Answer true if anElement is a member
  UObject GetFirst(out long rPosition); // Get the first element, set rPosition to
                                        // next position

  UObject GetNext(inout long rPosition); // Get element at rPosition, set rPosition to
                                        // the next one
   };

   // UObSet
   // A UObSet is a UObBag with the constraint that there are no duplicate elements.

   interface UObSet : UObBag
   {
      void Add_Unique(in UObject anElement);        // Add anElement to the set
   };

};

#endif
```

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

E.3 TFF_Port

Table E.3 contains the IDL for TFF_Port.

Table E.3—TFF_Port module represented in IDL

```
#ifndef _TFFPORT_IDL
#define _TFFPORT_IDL

#include "stdtypes.idl"

module TFF_Port
{
   interface Port : UObject
   {};
};

#endif
```

E.4 TFF_Connection

Table E.4 contains the IDL for TFF_Connection.

Table E.4—TFF_Connection module represented in IDL

```
#ifndef _TFFCONN_IDL
#define _TFFCONN_IDL

#include "stdtypes.idl"
#include "collect.idl"
#include "tffport.idl"

module TFF_Connection
{
   interface Connection : UObject
   {
      readonly attribute TFF_Port::Port port1;
           // The location name of the UUT end of the connection.

      readonly attribute TFF_Port::Port port2;
           // The location name of the Test System end of the
           // connection.

      void define(in TFF_Port::Port p1, in TFF_Port::Port p2);
            // This creates a connection by setting the attributes,
           // port1 = p1, port2 = p2.
   };

   interface Connection_UObBag : Collection::UObBag {};
   interface Connection_UObSet : Collection::UObSet {};

   interface ConnectionSet : UObject
   {
      attribute Connection_UObSet connectionData;
            // This set holds all of the connections that are to be
            // active simultaneously.

      void define(in Connection_UObSet connectionData);
            // Initialize the instance.

   };
};

#endif
```


Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

E.5 TFF_Resource

Table E.5 contains the IDL for TFF_Resource.

Table E.5—TFF_Resource module represented in IDL

```
#ifndef _TFFRES_IDL
#define _TFFRES_IDL

#include "stdtypes.idl"

module TFF_Resource
{

// The Resource interface includes services to implement the methods delegated
// to it from Signal or TestProperty. For this to work properly, the Resource
// instance must save the object references with which it is initialized.

   interface Resource : UObject
   {};
};

#endif
```

E.6 TFF_TestProperty

Table E.6 contains the IDL for TFF_TestProperty.

Table E.6—TFF_TestProperty module represented in IDL

```
#ifndef _TESTPROP_IDL
#define _TESTPROP_IDL

#include "stdtypes.idl"

module TFF_TestProperty
{
   exception E_HIGHLOWORDER {};
   exception E_OUTOFRANGE {};

// The FloatValueInterval object is used to provide functionality for intervals
// such as ranges and limits.

   interface FloatValueInterval
   {
      attribute float lowValue;
         // The low value of the interval

      attribute float highValue;
         // The high value of the interval

      void setInterval(in float lowValue, in float highValue)
           raises(E_HIGHLOWORDER);
         // Sets the interval endpoints, low from lowValue, high from
         // highValue

      InInterval testInterval(in float t);
         // The testInterval method answers as follows:
         //    LOW:  if t < low
         //    IN:   if low <= t <= high
         //    HIGH: if t > high
   };

   interface IntegerValueInterval
```

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Table E.6—TFF_TestProperty module represented in IDL *(continued)*

```
   {
      attribute long lowValue;
         // The low value of the interval

      attribute long highValue;
         // The high value of the interval

      void setInterval(in long lowValue, in long highValue)
           raises(E_HIGHLOWORDER);
         // Sets the interval endpoints, low from lowValue, high from
         // highValue

      InInterval testInterval(in long t);
         // The testInterval method answers as follows:
         //    LOW:  if t < low
         //    IN:   if low <= t <= high
         //    HIGH: if t > high
   };

// TestProperty class
// This class is the superclass of all TestProperty types.

   interface TestProperty : UObject
   {
      attribute NameType propType;
         // The propType field specifies the type of TestProperty. This is
         // actually a class name, and could be implemented like here as a
         // string, or as an enumeration, or some other way. It is included
         // here because the objects which use Behavior need to know the
         // propType value. (?)

      readonly attribute UObject partOf;
         // This object reference specifies which object the TestProperty is
         // a part of. This could be either a Signal, a Resource.

      void initialize(in UObject owner);
         // Sets the partOf variable to the owner of this property.
   };

   interface FloatTestProperty : TestProperty
   {
      attribute boolean measurement;
         // This attribute is set to TRUE when the TestProperty instance is to
         // be measured as part of a test sequence.

      attribute FloatValueInterval range;
         // The range of required or allowable values for the TestProperty.
         // Range is considered to be not set if either range = nil or
         // range.low = range.high = 0.

      attribute float accuracy;
         // A number specifying the accuracy at which the TestProperty value
         // is defined.

      readonly attribute float value;
         // The current value of the TestParameter. The value is maintained in
         // base (unscaled) units.

      attribute Engineering_Units units;
        // The units here are either base SIUnits or derived therefrom.

      void setValue(in float v) raises(E_OUTOFRANGE);
         // Sets the current TestProperty value to that of the argument.
         // If the value is out of range, the E_OUTOFRANGE is raised.
         // if no range is specified, the E_OUTOFRANGE exception is never raised.
   };

   interface IntegerTestProperty : TestProperty
   {
```


Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

Table E.6—TFF_TestProperty module represented in IDL *(continued)*

```
      attribute boolean measurement;
         // This attribute is set to TRUE when the TestProperty instance is to
         // be measured as part of a test sequence.

      attribute IntegerValueInterval range;
         // The range of required or allowable values for the TestProperty.
         // Range is considered to be not set if either range = nil or
         // range.low = range.high = 0.

      attribute long accuracy;
         // A number specifying the accuracy at which the TestProperty value
         // is defined.

      readonly attribute long value;
         // The current value of the TestParameter. The value is maintained in
         // base (unscaled) units.

      attribute Engineering_Units units;
         // The units here are either base SIUnits or derived therefrom.

      void setValue(in long v) raises(E_OUTOFRANGE);
         // Sets the current TestProperty value to that of the argument.
         // If the value is out of range, the E_OUTOFRANGE is raised.
         // if no range is specified, the E_OUTOFRANGE exception is never raised.
   };

   interface StringProperty : TestProperty
   {
      attribute string value;
         // The string specifying the value of the StringProperty.
   };
};

#endif
```

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

E.7 TFF_Behavior

Table E.7 contains the IDL for TFF_Behavior.

Table E.7—TFF_Behavior module represented in IDL

```
#ifndef _BEHAVIOR_IDL
#define _BEHAVIOR_IDL

#include "stdtypes.idl"
#include "collect.idl"
#include "tfftprop.idl"

module TFF_Behavior
{
   interface TestProperty_UObBag : Collection::UObBag {};
   interface TestProperty_UObSet : Collection::UObSet {};

   enum SType {SOURCE,SENSOR,LOAD};

   interface Behavior : UObject
   {
      attribute NameType objType;
         // The type of the object described by the behavior.  This is really a
         // class name, and in some implementations, this could be an enumeration
         // rather than a string.

      readonly attribute TestProperty_UObSet properties;
         // The set of TestProperties which define the behavior.

   };

   interface Behavior_UObBag : Collection::UObBag {};
   interface Behavior_UObSet : Collection::UObSet {};
};


// Additional comments.  Note that all of the instance variables can be read, and
// three of them can be written as attributes.  This is because any instance of a
// behavior object is built by either the Signal or resource it describes.

#endif
```


--`,,`-`-`,,`,,`,`,,`---
Copyright © 1998 IEEE. All rights reserved.
Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS
Not for Resale

E.8 TFF_Signal

Table E.8 contains the IDL for TFF_Signal.

Table E.8—TFF_Signal module represented in IDL

```
#ifndef _SIGNAL_IDL
#define _SIGNAL_IDL
#include "stdtypes.idl"
#include "tffconn.idl"
#include "tfftprop.idl"
#include "tffres.idl"

module TFF_Signal
{

   interface Signal : UObject
   {
      attribute TFF_Resource::Resource implementor;
         // The Resource object which implements the signal.

      attribute Signal trigger;
         // The signal that triggers operations on this one.

      attribute TFF_Connection::ConnectionSet connections;
         // The set of connections which must be satisfied
         // simultaneously when this signal is used.

      void initialize(in NameType aName);
         // Initializes the instance, and sets its name.
   };
};

#endif
```

Copyright © 1998 IEEE. All rights reserved.

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS Not for Resale

To order IEEE standards…

Call 1. 800. 678. IEEE (4333) in the US and Canada.

Outside of the US and Canada:
1. 732. 981. 0600

To order by fax:
1. 732. 981. 9667

IEEE business hours: 8 a.m.–4:30 p.m. (EST)

For on-line access to IEEE standards information…

Via the World Wide Web:
http://standards.ieee.org/

Via ftp:
stdsbbs.ieee.org

Via a modem:
1. 732. 981. 0035

ISBN 0-7381-0180-X

--`,-`-`,,`,,`,`,`,,`---

Copyright The Institute of Electrical and Electronics Engineers, Inc.
Provided by IHS under license with IEEE
No reproduction or networking permitted without license from IHS

Not for Resale