Distributed Simulation Systems Specification

November 2002
Version 2.0
formal/02-11-11

An Adopted Specification of the Object Management Group, Inc.

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF

MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE.
IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 250 First Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

The OMG Object Management Group Logo®, CORBA®, CORBA Academy®, The Information Brokerage®, XMI® and IIOP® are registered trademarks of the Object Management Group. OMG™, Object Management Group™, CORBA logos™, OMG Interface Definition Language (IDL)™, The Architecture of Choice for a Changing World™, CORBAservices™, CORBAfacilities™, CORBAmed™, CORBAnet™, Integrate 2002™, Middleware That's Everywhere™, UML™, Unified Modeling Language™, The UML Cube logo™, MOF™, CWM™, The CWM Logo™, Model Driven Architecture™, Model Driven Architecture Logos™, MDA™, OMG Model Driven Architecture™, OMG MDA™ and the XMI Logo™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

ISSUE REPORTING

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents & Specifications, Report a Bug/Issue.

Contents

Preface

About This Document

Under the terms of the collaboration between OMG and The Open Group, this document is a candidate for adoption by The Open Group, as an Open Group Technical Standard. The collaboration between OMG and The Open Group ensures joint review and cohesive support for emerging object-based specifications.

Object Management Group

The Object Management Group, Inc. (OMG) is an international organization supported by over 600 members, including information system vendors, software developers and users. Founded in 1989, the OMG promotes the theory and practice of object-oriented technology in software development. The organization's charter includes the establishment of industry guidelines and object management specifications to provide a common framework for application development. Primary goals are the reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments. Conformance to these specifications will make it possible to develop a heterogeneous applications environment across all major hardware platforms and operating systems.

OMG's objectives are to foster the growth of object technology and influence its direction by establishing the Object Management Architecture (OMA). The OMA provides the conceptual infrastructure upon which all OMG specifications are based. More information is available at http://www.omg.org/.

The Open Group

The Open Group, a vendor and technology-neutral consortium, is committed to delivering greater business efficiency by bringing together buyers and suppliers of information technology to lower the time, cost, and risks associated with integrating new technology across the enterprise.

The mission of The Open Group is to drive the creation of boundaryless information flow achieved by:

- Working with customers to capture, understand and address current and emerging requirements, establish policies, and share best practices;
- Working with suppliers, consortia and standards bodies to develop consensus and facilitate interoperability, to evolve and integrate specifications and open source technologies;
- Offering a comprehensive set of services to enhance the operational efficiency of consortia; and
- Developing and operating the industry's premier certification service and encouraging procurement of certified products.

The Open Group has over 15 years experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification. The Open Group portfolio of test suites includes tests for CORBA, the Single UNIX Specification, CDE, Motif, Linux, LDAP, POSIX.1, POSIX.2, POSIX Realtime, Sockets, UNIX, XPG4, XNFS, XTI, and X11. The Open Group test tools are essential for proper development and maintenance of standards-based products, ensuring conformance of products to industry-standard APIs, applications portability, and interoperability. In-depth testing identifies defects at the earliest possible point in the development cycle, saving costs in development and quality assurance.

More information is available at http://www.opengroup.org/ .

OMG Documents

The OMG Specifications Catalog is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

The OMG documentation is organized as follows:

OMG Modeling Specifications

Includes the UML, MOF, XMI, and CWM specifications.

OMG Middleware Specifications

Includes CORBA/IIOP, IDL/Language Mappings, Specialized CORBA specifications, and CORBA Component Model (CCM).

Platform Specific Model and Interface Specifications

Includes CORBAservices, CORBAfacilities, OMG Domain specifications, OMG Embedded Intelligence specifications, and OMG Security specifications.

Obtaining OMG Documents

The OMG collects information for each book in the documentation set by issuing Requests for Information, Requests for Proposals, and Requests for Comment and, with its membership, evaluating the responses. Specifications are adopted as standards only when representatives of the OMG membership accept them as such by vote. (The policies and procedures of the OMG are described in detail in the *Object Management Architecture Guide*.) OMG formal documents are available from our web site in PostScript and PDF format. Contact the Object Management Group, Inc. at:

OMG Headquarters

250 First Avenue

Needham, MA 02494

USA

Tel: +1-781-444-0404

Fax: +1-781-444-0320

pubs@omg.org

http://www.omg.org

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Helvetica bold - OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier bold` - Programming language elements.

Helvetica - Exceptions

Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Acknowledgments

The following company submitted this CORBA Manufacturing specification:

- Defense Modeling and Simulation Office (DMSO)

DSS Specification 1

Contents

This chapter contains the following sections.

1.1 Introduction

The Facility for Distributed Simulation Systems is a mechanism for combining computer models or simulations so that they interoperate to create one larger simulation.

This facility is equivalent to the High Level Architecture (HLA) for modeling and simulation that originated in the U.S. Department of Defense, and was later adopted by the Institute of Electrical and Electronics Engineers (IEEE). The purpose of this specification is to incorporate the relevant portions of the IEEE HLA specification and to define suitable CORBA IDL interfaces.

The following terms are used throughout this specification:

- The combined simulation system created from its constituent simulations is a *federation.*
- Each simulation that is combined to form a federation is called a *federate.*

- A *federation execution* is a session of a federation executing together.

A federation consists of several elements:

- Some number of federates
- An implementation of the *Runtime Infrastructure* (RTI) defined by this specification (one RTI may serve several federations, but each federation has one RTI)
- A *Federation Object Model* (FOM) that defines, as an M1 model, the information exchanged between federates through the RTI in the federation (one FOM may serve several federations, but each federation has one FOM).

This is illustrated in Figure 1-1.

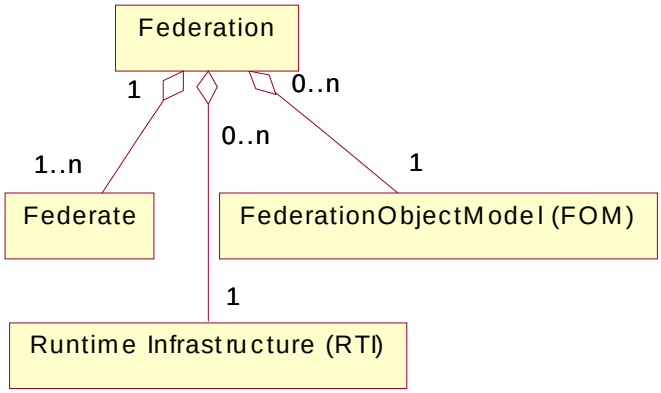

Figure 1-1 Constituents of a Federation

The specification concerns itself with two things: the interface between each federate and the RTI, and the structure of each FOM. The interface is specified abstractly as services in the IEEE specification. The relevant portions of that specification are incorporated here, and this specification defines CORBA IDL interfaces that represent the IEEE services. The structure of each FOM is controlled by a meta-model (M2) called the Object Model Template (OMT). The OMT is defined in a related IEEE specification, incorporated here. At federation execution time the FOM is represented to the RTI in the form of the *FOM Document Data* (FDD). The content and structure of this document is contained in the IEEE specification.

Typically in a federation several federates are connected to one RTI. Each federate invokes operations on an instance of a CORBA interface called **RTIambassador**. The RTI must invoke services on each federate as well. Each federate presents to the RTI an instance of the interface **FederateAmbassador**, on which the RTI invokes services meant for that federate.

This is illustrated in Figure 1-2.

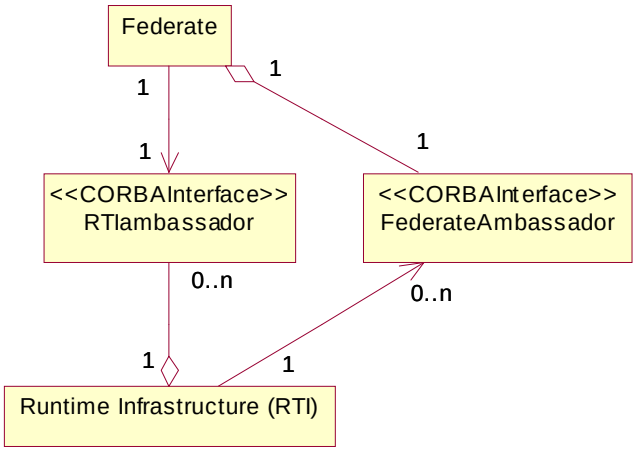

Figure 1-2 Interface Between Federates and the RTI

1.2 Specifications Incorporated by Reference

The following specifications are hereby incorporated by reference:

- 1516.1-2000 IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Federate Interface Specification, clauses 1 through 11 only
- 1516.2-2000 IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Object Model Template (OMT) Specification

Note – IEEE 1516.1 is hereafter referred to as the *Interface Specification*. IEEE 1516.2 is hereafter referred to as the *OMT Specification*.

This specification is intended to be read only after the Interface and OMT Specifications have been read; in particular the mappings to CORBA IDL in this specification should be read after the service group overviews and language-neutral service definitions in the Interface Specification. The HLA is a complex architecture; the most comprehensive tutorial introduction to the HLA is the following:

Kuhl, Frederick.; Weatherly, Richard; and Dahmann, Judith: *Creating Computer Simulation Systems, an Introduction to the High Level Architecture*. Prentice Hall, 2000.

1.3 Mapping to CORBA IDL

The services defined in the Interface Specification (clauses 4 through 11) are presented in language-neutral terms. The purpose of this section is to describe the mapping from the language-neutral service definitions to the corresponding CORBA IDL interfaces.

Federate-initiated services correspond to methods on the interface **RTIambassador**. RTI-initiated services, which are marked in the Interface Specification with a printer's dagger (†), correspond to methods on the interface **FederateAmbassador**. RTI-initiated services are also called *callbacks*.

RTI implementers shall furnish an implementation of **RTIambassador**. Federate implementers shall furnish an implementation of **FederateAmbassador**.

1.3.1 Services

In general, each service defined in the Interface Specification (clauses 4 through 11) corresponds to one method on the interfaces **RTIambassador** or **FederateAmbassador**. In some cases, because of optional arguments to a service or programming language constraints, a single service may be mapped to multiple methods.

1.3.1.1 Join federation execution

In addition to the supplied arguments listed with the description of this service, the IDL API requires the federate to supply an instance of **FederateAmbassador** and instances of **LogicalTimeFactory** and **LogicalTimeIntervalFactory**.

1.3.1.2 Register federation synchronization point

The optional argument representing the set of joined federate designators is implemented as two **registerFederationSynchronizationPoint** methods, one that takes a **FederateHandleSet** as an argument, and one that does not.

1.3.1.3 Confirm synchronization point registration †

The registration success indicator is implemented as two distinct methods: **synchronizationPointRegistrationSucceeded()** and **synchronizationPointRegistrationFailed()**. A Synchronization Point Failure Reason (discussed in 2.3.2.9), in the form of a **SynchronizationPointFailureReason**, is passed as an argument with **synchronizationPointRegistrationFailed()**.

1.3.1.4 Request federation save

The optional time stamp argument on this service is implemented as two methods, one of which takes a **LogicalTime** as an argument.

1.3.1.5 Initiate federate save †

The optional time stamp argument on this service is implemented as two methods, one of which takes a **LogicalTime** as an argument.

1.3.1.6 Federate save complete

This service corresponds to two methods: **federateSaveComplete()** and **federateSaveNotComplete()**.

1.3.1.7 Federation saved †

This service corresponds to two methods: **federationSaved()** and **federationNotSaved()**. A Save Failure Reason (discussed in 2.3.2.9), in the form of a **SaveFailureReason**, is passed as an argument to **federationNotSaved()**.

1.3.1.8 Confirm federation restoration request †

This service is mapped to the following two methods: **requestFederationRestore Succeeded()** and **requestFederationRestoreFailed()**.

1.3.1.9 Federate restore complete

This service is mapped to the following two methods: **federateRestoreComplete()** and **federateRestoreNotComplete()**.

1.3.1.10 Federation restored †

This service corresponds to two methods: **federationRestored()** and **federationNotRestored()**. A Restore Failure Reason (discussed in 2.3.2.9), in the form of a **RestoreFailureReason**, is passed as an argument to **federationNotRestored()**.

1.3.1.11 Unpublish object class attributes

The optional argument representing the set of attribute designators is implemented as two methods: **unpublishObjectClass()** and **unpublishObjectClassAttributes()**. The latter takes an **AttributeHandleSet** as an argument.

1.3.1.12 Subscribe object class attributes

The optional passive subscription indicator is implemented as two methods: **subscribe ObjectClassAttributes()** and **subscribeObjectClassAttributesPassively()**.

1.3.1.13 Unsubscribe object class attributes

The optional argument representing the set of attribute designators is implemented as two methods: **unsubscribeObjectClass()** and **unsubscribeObjectClassAttributes()**. The latter of the two takes an **AttributeHandleSet** as an argument.

1.3.1.14 Subscribe interaction class

The optional passive subscription indicator is implemented as two methods: **subscribeInteractionClass()** and **subscribeInteractionClassPassively()**.

1.3.1.15 Object instance name reserved †

This service corresponds to two methods: **objectInstanceNameReservationSucceeded()** and **objectInstanceNameReservationFailed()**.

1.3.1.16 Register object instance

The optional argument representing the object instance name is implemented as two **registerObjectInstance()** methods, one that takes a **wstring** as an argument for the object instance name, and one that does not.

1.3.1.17 Update attribute values

The optional argument representing the time stamp is implemented as two **uptdateAttributeValues()** methods, one that takes a **LogicalTime** as an argument for the time stamp, and one that does not. The version that takes a **LogicalTime** also returns a **MessageRetractionHandle** for the optional message retraction designator, along with a **boolean out** parameter indicating whether the handle is valid.

1.3.1.18 Reflect Attribute Values †

This service corresponds to six overloaded versions of the **reflectAttributeValues()** method. The first version takes none of the optional arguments. The second version adds only the optional set of sent region designators, represented as an instance of a **RegionHandleSet**. Each of the remaining four versions takes a **LogicalTime** as an argument for the time stamp and takes an **OrderType** as an argument for the receive message order type. To this, the fourth version adds the optional set of sent region designators, represented as an instance of a **RegionHandleSet**. The fifth and sixth versions take a **MessageRetractionHandle** as an argument for the message retraction designator. The sixth version adds the optional set of sent region designators, represented as an instance of a **RegionHandleSet**.

1.3.1.19 Send interaction

The optional argument representing the time stamp is implemented as two **sendInteraction()** methods, one that takes a **LogicalTime** as an argument for the time stamp, and one that does not. The version that takes a **LogicalTime** also returns a **MessageRetractionHandle** for the optional message retraction designator, along with a **boolean out** parameter indicating whether the handle is valid.

1.3.1.20 Receive interaction †

This service corresponds to six overloaded versions of the **receiveInteraction()** method. The first version takes none of the optional arguments. The second version adds only the optional set of sent region designators, represented as an instance of a **RegionHandleSet**. Each of the remaining four versions takes a **LogicalTime** as an argument for the time stamp and takes an **OrderType** as an argument for the receive message order type. To these the fourth version adds the optional set of sent region designators, represented as an instance of a **RegionHandleSet**. The fifth and sixth versions take a **MessageRetractionHandle** as an argument for the message retraction designator. The sixth version adds the optional set of sent region designators, represented as an instance of a **RegionHandleSet**.

1.3.1.21 Delete object instance

The optional argument representing the time stamp is implemented as two **deleteObjectInstance()** methods, one that takes a **LogicalTime** as an argument for the time stamp, and one that does not. The version that takes a **LogicalTime** also returns a **MessageRetractionHandle** for the optional message retraction designator, along with a **boolean out** parameter indicating whether the handle is valid.

1.3.1.22 Remove object instance †

This service corresponds to three overloaded versions of the **removeObjectInstance()** method. The first version takes none of the optional arguments. Both the remaining two versions take a **LogicalTime** as an argument for the time stamp, and take an **OrderType** as an argument for the receive message order type. Finally, the third version takes a **MessageRetractionHandle** as an argument for the message retraction designator.

1.3.1.23 Request attribute value update

This service corresponds to two overloaded methods named **requestAttributeValueUpdate()**. One takes an **ObjectInstanceHandle** to represent the object instance designator. The other takes an **ObjectClassHandle** to represent the object class designator.

1.3.1.24 Inform attribute ownership †

This service corresponds to three distinct methods: **attributeIsNotOwned()**, **attributeIsOwnedByRTI()**, and **informAttributeOwnership()**. All three methods take an **ObjectInstanceHandle** to represent the object instance designator and an **AttributeHandle** to represent the attribute designator as arguments. In addition, **informAttributeOwnership()** takes a **FederateHandle** as an argument to represent the federate designator.

1.3.1.25 Register object instance with regions

This service corresponds to two overloaded versions of the **registerObjectInstanceWithRegions()** method. The second version takes a **wstring** as an argument to represent the optional object instance name.

1.3.1.26 Subscribe object class attributes with regions

The optional passive subscription indicator is implemented as two methods: **subscribeObjectClassAttributesWithRegions()** and **subscribeObjectClassAttributesPassivelyWithRegions()**.

1.3.1.27 Subscribe interaction class with regions

The optional passive subscription indicator is implemented as two methods: **subscribeInteractionClassWithRegions()** and **subscribeInteractionClassPassivelyWithRegions().**

1.3.1.28 Send interaction with regions

The optional argument representing the time stamp is implemented as two **sendInteractionWithRegions()** methods, one that takes a **LogicalTime** as an argument for the time stamp, and one that does not. The version that takes a **LogicalTime** also returns a **MessageRetractionHandle** for the optional message retraction designator, along with a **boolean out** parameter indicating whether the handle is valid.

1.3.2 Data Types

1.3.2.1 Names, labels, and strings

The names, labels, and strings described in clauses 4 through 10 of the Interface Specification are mapped to **wstring**.

1.3.2.2 Boolean values

The IDL type **boolean** is used to represent all Boolean values.

1.3.2.3 Exceptions

Exceptions listed in clauses 4 through 10 of the Interface Specification map to IDL exceptions.

1.3.2.4 Handles

As explained in 1.4.2 of the Interface Specification, the Interface Specification uses the term “designator” as a generalization of names and handles. Most services require handles. Clause 10 of the Interface Specification presents the services that are used to map names to handles and handles to names.

Names in most cases (e.g., object class names, attribute names, interaction class names, etc.) come from the FDD. Thus they are known ahead of time to all federates. Object instance names, however, are different. These names may be chosen by the registering federate, or they may be created dynamically by the RTI. Thus these names often are not predictable.

Handles in all cases are generated by the RTI. They are intended to be compact tokens, assigned by the RTI, used for the sake of efficiency in place of names. Where the handles correspond to names from the FDD, these handles shall be generated by the RTI repeatably. This means that if two federation executions are created on the same version of the same RTI with the same FDD file, then the handles assigned to the same name shall be the same in each federation execution. This property facilitates saves and restores in that handles, rather than the names, can be saved by federates. This property does not, however, imply that an RTI must assign handles in a manner known *a priori* to federates.

Handles shall be unique in a federation execution. Each federate shall know the same item by the same handle. This means that the federate registering an object instance shall know it by the same handle as another federate that discovers the object instance.

Each kind of handle is represented in IDL as an **interface** (e.g., **ObjectClassHandle** or **FederateHandle**). An RTI implementer will implement the interface for each such interface, the details of which are hidden from the federate developer.

The methods on **RTIambassador** that return handles will return instances of the corresponding implementer-provided interface. The RTI user, i.e., a federate developer, cannot see, nor should be concerned with, the RTI implementer’s implementation of the interface.

The consequences of this scheme are that, firstly, handle types are type-safe, and, secondly, RTI developers are free to adopt any form of internal state for handles that they choose.

Each handle interface supports **equals()** and **hash_code()** so that handles can be used as keys in hash tables.

Often during a federation execution, it is useful for federates to send information identifying an item to another federate. For example, one federate may wish to send an interaction to another federate identifying a particular object instance. To send information identifying an item to another federate, there are two options: send the name of the item, or send the handle of the item.

All data sent between federates in an HLA federation are sent as attribute or parameter values. To send a handle as an attribute value or as a parameter value, the handle must be encoded before it is sent, and it must be decoded and reconstituted by the receiver. (Note that we speak here of sending a handle from one federate to another as simulation data. In the case of any **RTIambassador** or **FederateAmbassador** service, where a handle is conveyed between a federate and the RTI as a service parameter, the handle is represented as a **interface**, and the usual CORBA marshaling is applied.)

Each handle interface supports **encode()**, which will create a compact, opaque representation suitable for network transmission in a returned octet sequence. Each handle interface has a corresponding handle factory interface, an implementation of which must also be furnished by the RTI implementer. The factory interface defines **decode()**, which returns a new handle instance from a representation in the provided octet sequence. This mechanism is intended to allow handles to be saved and restored later, as well as to allow handles to be passed among federates as attribute values or parameter values.

A handle of a given type (e.g., an object instance handle) shall only be reconstitutable as that type.

Handles that are sent as part of MOM attribute value updates or as parameter values of MOM interactions are sent encoded and may be reconstituted in the manner described above.

Handle types also provide a **to_string()** method that returns a printable form of a handle. The string produced, however, is not guaranteed to be the same as the name to which the handle corresponds (assuming there is a name that corresponds to that kind of handle). If the user needs the name form of a handle, the appropriate support service from clause 10 of the Interface Specification should be used.

1.3.2.5 *Resignation directive*

The *Resign Federation Execution* service allows the specification of one of six directives. These are represented by the **enum ResignAction**.

1.3.2.6 *Attribute and parameter values and user-supplied tag*

Attribute and parameter values and user-supplied tags are represented, respectively, as **AttributeValue**, **ParameterValue**, and **UserSuppliedTag**. These are all **octet** sequences.

1.3.2.7 *Sets of designators*

Sets of designators, e.g., federate designators in *Register Federation Synchronization Point* or attribute designator sets in *Publish Object Class Attributes*, are represented by sequences of the corresponding handles. Handle sets are provided for federate handles, attribute handles, dimension handles, and region handles.

1.3.2.8 Success indicators

Several of the requests a federate may make of an RTI must be acknowledged by the RTI to the federate. This acknowledgment can indicate that the request succeeded or failed. Each acknowledgment corresponds to positive and a negative callback methods on the **FederateAmbassador**. Thus, for example, the following pairs: **federationSaved()** and **federationNotSaved()**; **requestFederation RestoreSucceeded()** and **requestFederationRestoreFailed()**; and **federation Restored()** and **federationNotRestored()**.

1.3.2.9 Failure reasons

Reasons for failure of an operation, as returned by **FederateAmbassador** methods **synchronizationPointRegistrationFailed(), federationNotSaved(),** and **federationNotRestored()** are represented by IDL **enum**s **SynchronizationPointFailureReason, SaveFailureReason**, and **RestoreFailureReason**, respectively.

1.3.2.10 Save and restore status

The status returned for each federate by the callbacks **federationSaveStatusResponse()** and **federationRestoreStatusResponse()** is represented by **enum**s **SaveStatus** and **RestoreStatus**, respectively. These methods return sequences of **struct**s, each of which contains a **FederateHandle** and a status **enum**.

1.3.2.11 Passive subscription indicator

The *Subscribe Object Class Attributes*, *Subscribe Interaction Class*, *Subscribe Object Class Attributes With Region*s, and *Subscribe Interaction Class With Regions* services each may optionally provide a Boolean value to indicate whether or not the subscription is active. Each such service corresponds to a pair of methods on **RTIambassador**, e.g., **subscribeObjectClassAttributes()** and **subscribeObjectClassAttributesPassively()**.

1.3.2.12 Constrained set of attribute designator and value pairs

The *Update Attribute Values* and *Reflect Attribute Values †* services require as arguments a set of attribute designator and value pairs. Such sets are instances of **AttributeHandleValuePairSequence**. No such sequence passed as a service argument shall contain two pairs with the same **AttributeHandle**.

1.3.2.13 Constrained set of parameter designator and value pairs

The *Send Interaction*, *Receive Interaction †*, and *Send Interaction With Regions* services require as arguments a set of parameter designator and value pairs. Such sets are instances of **ParameterHandleValuePairSequence**. No such sequence passed as a service argument shall contain two pairs with the same **ParameterHandle**.

1.3.2.14 Message order and transportation types

Message order types are represented by an IDL **interface** called **OrderType**. A federate retrieves an **OrderType** value from the RTI through the *Get Order Type* service. Like handles, **OrderType**s can be compared and used as indexes. They can be encoded for transmission as attribute or parameter values. The **to_string()** method is not guaranteed to return the corresponding name.

Similar observations apply to the **TransportationType**.

1.3.2.15 Ownership designator

The *Inform Attribute Ownership †* service supplies an ownership designator as one of its arguments. This corresponds to three distinct methods: **attributeIsNotOwned()**, **attributeIsOwnedByRTI()**, and **informAttributeOwnership()**. All three methods carry an **ObjectInstanceHandle** and an **AttributeHandle** as arguments. In addition, **informAttributeOwnership()** carries a **FederateHandle** as an argument. Together, these three methods combine to represent the ownership designator of the *Inform Attribute Ownership †* service.

1.3.2.16 Definition indicator

The *Query GALT* and *Query LITS* services are required to return an indicator to define whether or not their return value is valid. Both services return a **boolean out** argument called **timeIsValid** that indicates whether the **LogicalTime** is valid. If the argument **timeIsValid** is **FALSE**, the argument **time** should not be used.

1.3.2.17 Optional message retraction designator

The *Update Attribute Value*s, *Send Interaction*, *Send Interaction With Regions*, and *Delete Object Instance* services may return a message retraction designator under certain circumstances.The methods corresponding to these services return a **boolean out** argument called **retractionHandleIsValid** that indicates whether the retraction handle is valid. If the argument **retractionHandleIsValid** is **FALSE**, the argument **handle** should not be used.

1.3.2.18 Collection of attribute designator set and region designator set pairs

Several services require the federate to provide the RTI with a collection of attribute designator set and region designator set pairs. This corresponds to **AttributeSetRegionSetPairSequence**. No such sequence passed as a service argument shall contain two pairs with the same **AttributeHandleSet**.

1.3.2.19 Logical time, time stamps, and lookahead

1.3.2.20 Logical time and logical time intervals are represented by a double value.

1.3.2.21 Dimension upper bound and range lower and upper bounds

Several DDM-related services allow bounds to be get and set. The *Get Dimension Upper Bound* service gets the upper bound of a particular dimension. The *Get Range Bounds* and *Set Range Bounds* services are used to get and set the bounds of a range of a region. All bounds are represented in IDL as a **long long**. The *Get Dimension Upper Bound* service, as it is getting a single bound value, returns a **long long**. The *Get Range Bounds* and *Set Range Bounds* services, as they are getting and setting a pair of values, return and take a **RangeBounds struct**, which is composed of a pair of **long long**s.

1.3.2.22 Wall-clock time

The *Evoke Callback* and *Evoke Multiple Callbacks* services take arguments specifying durations of wall-clock time expressed in seconds. Wall-clock time is represented using a **double**.

1.4 Interpretation of Specifications Incorporated by Reference

1.4.1 Introduction

Some areas of the IEEE Standard For Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification (IEEE Std 1516.1-2000) are not as well-specified as they could be and as a result may have questionable interpretations. Consequently, as a guide to RTI developers and users, we offer the following interpretations of its ambiguous portions. All services, section and clause numbers referred to in this section refer to parts of IEEE 1516.1-2000.

1.4.2 Definitions and Federation Management Interpretations

1.4.2.1 Definition 3.1.66: published

Interpretation 1

Part (a) of this definition reads, “pertaining to an object class such that, from the perspective of a given joined federate, there is at least one attribute of the object class that was an argument to a *Publish Object Class Attributes* service invocation that was not subsequently unpublished via the *Unpublish Object Class Attributes* service.” Part (a) of this definition should instead read (changes in boldface), “pertaining to an object class such that, from the perspective of a given joined federate, there is at least one

attribute **available at that object class** that, **along with the object class**, was an argument to a *Publish Object Class Attributes* service invocation that was not subsequently unpublished at that object class via the *Unpublish Object Class Attributes* service."

Rationale: Because it is possible to invoke the Publish Object Class service with an object class and an empty set of attributes, it should be made clear that an object class is not considered to be "published" unless at least one of the attributes available at that class was published along with the object class. An object class can only be considered to be published if at least one of the attributes available at that class was an argument to the Publish Object Class Attributes service along with the object class. A pre-condition of the Register Object Instance service is that "the joined federate is publishing the object class." Because of the definition of "published" above, this means that the federate must have invoked the Publish Object Class Attributes service for both that object class and for at least one available attribute of that object class. A federate that only invokes the Publish Object Class Attributes service with an object class and an empty set of attributes (without first invoking the Publish Object Class Attributes service with that object class and a non-empty set of attributes and not subsequently invoking the Unpublish Object Class Attributes service for that object class and those attributes) will not be allowed to register instances of that class.

1.4.2.2 Figure 3: Lifetime of a Federate

Interpretation 1

Many services in 1516.1 have preconditions and exceptions that are worded simply "Save in progress" or "Save not in progress." In all cases, the phrase "save in progress" is intended to refer to the "Federate Save in Progress" state as shown in the statechart in Figure 3. A save is considered to be in progress at a given federate if the federate is in the "Federate Save in Progress" state.

Interpretation 2

Many services in 1516.1 have preconditions and exceptions that are worded simply "Restore in progress" or "Restore not in progress." In all cases, the phrase "restore in progress" is intended to refer to the "Federate Restore in Progress" state as shown in the statechart in Figure 3. A restore is considered to be in progress at a given federate if the federate is in the "Federate Restore in Progress" state.

1.4.2.3 Service 4.4: Join Federation Execution

Interpretation 1

The introductory text says, "The returned joined federate designator shall be unique for the lifetime of the federation execution." However, it should say, "The returned joined federate designator shall be unique for the lifetime of the federation execution, as long as a restore is not in progress at any federate."

Rationale: During a restore operation, when the Initiate Federate Restore † service is invoked at a federate, one of the supplied arguments is a joined federate designator, with the result that the Initiate Federate Restore † service could cause a joined federate's designator to change from the value supplied by the Join Federation Execution service. This means that while a restore is in progress at one or more federates, it is possible that two different federates in the federation execution could have the same joined federate designator, one federate having the designator that was supplied to it by the Join Federation Execution service and one federate having the designator that was supplied to it by the Initiate Federate Restore † service. Therefore, the introductory text of the Join Federation Execution service is incorrect as written and needs to be amended as described in the interpretation above in order to be correct.

1.4.2.4 Service 4.5: Resign Federation Execution

Interpretation 1

If a federate invokes this service with either directive 1, 4, or 5, then for each instance attribute that becomes unowned as a result, if no joined federates are in either the "Acquiring" or "Willing to Acquire" state with respect to the specified instance attribute, the RTI shall try to identify other joined federates that are willing to own the instance attribute. If any joined federate is in either the "Acquiring" or "Willing to Acquire" state with respect to the specified instance attribute, the RTI may try to identify other joined federates that are willing to own the instance attribute. The mechanism that the RTI shall use to try to identify other joined federates that are willing to own the instance attribute is invocation of the Request Attribute Ownership Assumption † service at all joined federates that are both eligible to own the instance attribute and not in either the "Acquiring" or "Willing to Acquire" state with respect to the specified instance attribute.

Rationale: The Resign Federation Execution service description does not explain or even mention the mechanism by which the RTI is expected to try to find an owner for unowned instance attributes that become unowned as the result of invocation of the Resign Federation Execution service with directive 1, 4, or 5 (all of which involve ownership management). The text in this interpretation describes this mechanism. It makes clear the requirement that the RTI must try to find an owner for unowned instance attributes that become unowned as the result of the explicit use of an ownership management directive. If there are no federates that are trying to acquire an unowned instance attribute, then the RTI must use the Request Attribute Ownership Assumption † service as the mechanism for offering ownership of the unowned instance attribute to federates that are eligible to own it. If there is one or more federate that is trying to acquire an unowned instance attribute, then the RTI may either give ownership of the instance attribute to one of the federates that are trying to acquire it without offering ownership of it to other eligible federates, or it may use the Request Attribute Ownership Assumption † service to offer ownership of the unowned instance attribute to eligible federates before granting ownership of the attribute to a federate that expresses an interest in acquiring it.

1.4.2.5 Service 4.24: Query Federation Restore Status

Interpretation 1

The behavior of the Query Federation Restore Status service and its companion Federation Restore Status Response † service will not be tested.

Rationale: When a federate receives an Initiate Federate Restore † callback, one of the supplied arguments of this service is a joined federate designator. So, as a result of the invocation of the Initiate Federate Restore † service, a joined federate's designator could change. This means that while one or more federate is in the Federate Restore In Progress state, it is possible that two different federates could have the same federate ID: one federate having its pre-restore ID and the other federate having its post-restore ID. Given that a precondition of the invocation of the Query Federation Restore Status service is that a restore is in progress, it is not clear what list of joined federate designators would be reported while a restore is in progress, the pre-restore IDs, the post-restore IDs, or a mixture. It is not clear that any meaningful information about the list of joined federates in the federation execution could be reported via the Federation Restore Status Response † service while a restore is in progress.

1.4.3 Declaration Management Interpretations

1.4.3.1 Figure 10: Class Attribute (i,j)

Interpretation 1

The history transition on the right side of this diagram should be labeled "Subscribe(i,{}) or Unsubscribe (i,{})," as opposed to its current labeling of "Publish (i,{}) or Unpublish(i,{})."

1.4.3.2 Figure 11: Class Attribute (i, HLAprivilegeToDeleteObject)

Interpretation 1

The history transition on the right side of this diagram should be labeled "Subscribe(i,{}) or Unsubscribe (i,{})," as opposed to its current labeling of "Publish (i,{}) or Unpublish(i,{})."

1.4.3.3 Figure 12: Interaction Class (m)

Interpretation 1

On the left hand side of this statechart, the transitions between the "Unpublished (m)" state and the "Published (m)" state should read "Publish (m)" and "Unpublish (m)," rather than simply "Publish" and "Unpublish."

1.4.3.4 Service 5.2 Publish Object Class Attributes

Interpretation 1

The introductory text for this service says that one of the ways that a joined federate may become the owner of an instance attribute is:

- By using ownership management services to acquire instance attributes of object instances. The joined federate may acquire only those instance attributes for which the joined federate is publishing the corresponding class attributes.

To be more precise, this text should instead read"

- By using ownership management services to acquire instance attributes of object instances. The joined federate may acquire only those instance attributes for which the joined federate is publishing the corresponding class attributes **at the known class of the specified object instance**.

1.4.3.5 Service 5.3: Unpublish Object Class Attributes

Interpretation 1

Pre-condition (e) of this service implies that a federate could invoke the *Attribute Ownership Acquisition* service or the *Attribute Ownership Acquisition If Available* service on an instance attribute that the federate already owns. Pre-condition (e) says:

For each class attribute of the specified class that is published by the joined federate and is to be unpublished by this service invocation, there are no joined federate-owned corresponding instance attributes for which the joined federate has either

1. invoked the Attribute Ownership Acquisition service, and has not yet received a corresponding invocation of either the Confirm Attribute Ownership Acquisition Cancellation † service or the Attribute Ownership Acquisition Notification † service, or

2. invoked the Attribute Ownership Acquisition If Available service, and has not yet received a corresponding invocation of either the Attribute Ownership Unavailable † service or the Attribute Ownership Acquisition Notification † service, or

3. invoked the Attribute Ownership Acquisition If Available service and has subsequently invoked the Attribute Ownership Acquisition service [after which condition 1) applies].

The phrase, “joined federate-owned” should be deleted from the first sentence of above text, and replaced by text that refers instead to attributes that are unowned by the joined federate. The above pre-condition should instead read, “For each class attribute of the specified class that is published by the joined federate and is to be unpublished by this service invocation, there are no corresponding instance attributes that are unowned by this joined federate and for which the joined federate has either…”

1.4.3.6 Service 5.6: Subscribe Object Class Attributes

Interpretation 1

This service description needs to more completely describe the intended effect of the optional passive subscription indicator. It says, "If a subscription is provided for a class attribute that is already subscribed by the joined federate, then the subscription shall take on the effect of the optional passive subscription indicator from the most recent *Subscribe Object Class Attributes* service invocation." It does not say what should happen in the event that a subscription is provided for a class attribute that is not already subscribed. Furthermore, the text does not make clear whether the active/passive characteristic applies on a per-attribute or a per-class basis. The use of the optional passive subscription indicator is expected to work as follows:

> Each subscribed class attribute is subscribed either actively or passively (but not both actively and passively) at a given object class; and two class attributes that are subscribed at the same object class may be subscribed differently from each other: one active and one passive. Each class attribute specified in a given invocation of the *Subscribe Object Class Attributes* service will take on the effect of the optional passive/active subscription indicator supplied (or not supplied) with that service invocation. Invoking the *Subscribe Object Class Attributes* service with an empty set of class attributes shall not change the active/passive subscription nature of any of the attributes that are subscribed at the specified object class. Each use of the *Subscribe Object Class Attributes* service shall add to the subscriptions specified to the RTI in any previous *Subscribe Object Class Attributes* service invocation for the same object class and may change the active/passive nature of previous class attribute subscriptions for that object class.

Rationale: The expected behavior is for invocations of the Subscribe Object Class Attributes service for any given object class to be cumulative with respect to the set of attributes subscribed at that class, but substitutive with respect to whether each attribute is subscribed actively or passively. If the current invocation of the Subscribe Object Class Attributes service includes a given attribute as an argument, the property of active versus passive for that attribute is substituted according to the value (or absence) of the optional passive subscription indicator argument to the current invocation of the Subscribe Object Class Attributes service.

1.4.3.7 Service 5.10: Start Registration For Object Class †

Interpretation 1

Pre-condition (c) of this service should say, "At least one other joined federate in the federation execution is actively subscribed to at least one of the class attributes that the joined federate is publishing at the specified object class, and the object class at which the subscribing federate is actively subscribed to that attribute is the subscribing federate's candidate discovery class of an object instance registered at the specified object class."

Rationale: The purpose of the Start Registration for Object Class † service is to notify a publishing federate that registration of new object instances of the specified object class is advised. In other words, if only DM and not DDM is used in a federation execution, then it is the intention that receipt of the Start Registration For Object Class † service at a federate indicates to that federate that if it were to register an object instance at the specified class, then at least one other federate in the federation execution would discover that object instance and receive reflects for at least one instance attribute of that object instance. The way that pre-condition (c) is currently worded, this intention is not met because it specifies only that at least one other federate must be actively subscribed to at least one published attribute at the specified class or at a superclass of the specified class. This condition is not sufficient to ensure that the subscribing federate would discover the object instance. In fact, the subscribing federate must be subscribed to at least one published attribute at the ***candidate discovery class*** *of an object instance in order for the subscribing federate to discover the object instance.*

Consider the following example:

Fed1 publishes object class A.B (attribute X).

Fed2 subscribes to object class A.B (attribute Y) and to object class A(attribute X).

If Fed1 were to register an object instance of class A.B, Fed2 2 would not discover that object instance because Fed2 is not subscribed to attribute X (the only attribute that Fed1 is publishing) at the candidate discovery class of the object instance (class A.B). According to the way that pre-condition (c) is currently worded, however, Fed1 would receive a Start Registration For Object Class † service invocation for object class A.B. According to the way that pre-condition (c) would be worded under this interpretation, Fed1 would not receive a Start Registration For Object Class † service invocation.

1.4.3.8 Service 5.11: Stop Registration For Object Class †

Interpretation 1

Pre-condition (d) of this service should say, “None of the class attributes that the joined federate is publishing at the specified object class is actively subscribed to by any other joined federate in the federation execution at what would be the subscribing federate's candidate discovery class of an object instance registered at the specified object class.”

Rationale: The purpose of the Stop Registration for Object Class † service is to notify a publishing federate that registration of new object instances of the specified object class is not advised. In other words, it is the intention that receipt of the Stop Registration For Object Class † service at a federate indicates to that federate that if it were to register an object instance at the specified object class, then there is no other federate in the federation execution that would discover that object instance. Although the wording of pre-condition (d) meets this intention, it is overly restrictive, such that there could be instances in which registration of new object instances is not advised, even though pre-condition (d) is not met. As long as a subscribing federate is not subscribed to any of the published class attributes at the ***candidate discovery class,*** *the*

subscribing federate will not discover a registered object instance, even if the federate is subscribed to one or more of the published class attributes at a super-class of the specified class.

Consider the following example:

Fed1 publishes object class A.B (attribute X).

Fed2 subscribes to object class A.B (attributes X and Y).

Fed 1 receives a Start Registration For Object Class † callback for object class A.B.

Fed2 subscribes to object class A (attribute X).

Fed2 unsubscribes to object class A.B (attribute X) (so Fed2 is still subscribed to object class A.B (attribute Y).

If Fed1 were to register an object instance of class A.B, Fed2 would not discover that object instance because Fed2 is not subscribed to attribute X (the only attribute that Fed1 is publishing) at the candidate discovery class of the object instance (class A.B). According to the way that pre-condition (d) is currently worded, however, Fed1 would not receive a Stop Registration For Object Class † service invocation for object class A.B, because Fed2 is subscribed to attribute X at class A. According to the way that pre-condition (d) would be worded under this interpretation, Fed1 would receive a Stop Registration For Object Class † service invocation, because Fed2 is not subscribed to attribute X at the candidate discovery class of an object instance registered at class A.B.

1.4.4 Object Management Interpretations

1.4.4.1 Section 6.1: Overview of Object Management

Interpretation 1

According to this text,

An instance attribute of an object instance shall be in scope for joined federate F if

a) The object instance is known to the joined federate,

b) The instance attribute is owned by another joined federate, and

c) either

- The instance attribute's corresponding class attribute is a subscribed attribute of the known class of the object instance, or
- The instance attribute's corresponding class attribute is a subscribed attribute of the known class of the object instance with regions, and the update region set of the instance attribute overlaps the subscription region set of the instance attribute's corresponding class attribute at the known class of the instance attribute at the subscribing joined federate.

Item (b) above shall be changed to "The instance attribute is owned **either** by another joined federate **or by the RTI,** and".

Rationale: Note that there are three possible states of ownership of any given instance attribute: it may be owned by a federate, owned by the RTI, or not owned. Instance attributes of pre-defined attributes of MOM object instances are owned by the RTI, rather than being owned by another federate. This interpretation is required in order to allow a federate that is subscribed to a pre-defined class attribute of a MOM object class to receive the Attributes In Scope † callback for an instance attribute of a MOM object instance at that class. Without this interpretation, it would not be possible for any federates to receive Attributes In Scope † callbacks for any instance attributes corresponding to pre-defined MOM object class attributes.

1.4.4.2 Service 6.4: Register Object Instance

Interpretation 1

According to this service description, "the handle provided to the joined federate which registers the object instance shall be the same handle provided to all joined federates which discover the object instance."

The phrase "the same handle" is meant to refer to handle equality rather than handle identity. Two handles are considered to be the same if, according to the comparison operator in each of the APIs (for example, according to the "equals" method in the Java API) the handles would be determined to be equal. The handles must also have equality between federates that are using different language APIs. The handles may be communicated between federates via instance attributes or interaction parameters.

Rationale: The handles provided to each federate must be equal, but they do not have to be the same programming language object.

1.4.4.3 Service 6.5: Discover Object Instance

Interpretation 1

Pre-condition (d) of this service states that class attribute "*att*'s corresponding instance attribute that is part of the specified object instance is owned by another joined federate." This pre-condition should instead say that class attribute "*att*'s corresponding instance attribute that is part of the specified object instance is either owned by another joined federate or owned by the RTI." A corresponding change is also required in clause 6.1 where the conditions for the invocation of the *Discover Object Instance †* are defined. The first sub-bullet of item (b) in this definition should read, "either another joined federate (not F) or the RTI owns *i*, and".

Rationale: Note that there are three possible states of ownership of any given instance attribute: it may be owned by a federate, owned by the RTI, or not owned. Instance attributes of pre-defined attributes of MOM object instances are owned by the RTI, rather than being owned by another federate. This interpretation is required in order to

allow a federate that is subscribed to a pre-defined class attribute of a MOM object class to discover MOM object instances at that class. Without this interpretation, it would not be possible for any federates to discover MOM object instances.

1.4.4.4 Service 6.8: Send Interaction

Interpretation 1

The second sentence of this service description reads, “The interaction parameters shall only be those in the specified class and all super-classes, as defined in the FDD.” This statement shall be understood to mean that only parameters that are available at that interaction class may be sent in a given interaction, but a federate is not required to send all available parameters of the interaction class with the interaction.

1.4.4.5 Service 6.10: Delete Object Instance

Interpretation 1

It should be noted in conjunction with this service that the standard is silent regarding what will happen in the case in which a federate attempts to take ownership of an instance attribute of an object instance for which another federate has already scheduled a timed deletion. That is, if a federate schedules the deletion of an object instance for a time in the future, that object instance may still be discovered by other federates, and updates to instance attributes of that object instance may still be received by other federates, until their logical times are greater than or equal to the specified time of the deletion. If those other federates are allowed to take ownership of any of the instance attributes owned by the federate that scheduled the delete, then it would be possible for strange things to occur within the federation execution, such as a federate that owns the HLAprivilegeToDelete instance attribute of an object instance receiving a Remove Object Instance † callback for that object instance. The standard does not specify the RTI’s behavior in such circumstances.

1.4.5 Ownership Management Interpretations

1.4.5.1 Section 7.1: Overview of Ownership Management

Interpretation 1

Text in this section currently states, “The RTI shall be responsible for attempting to find an owner for instance attributes that are left unowned (either via registration, federate resignation, or some form of divestiture). The RID provided to an RTI may allow the federation to control how often or for how long an RTI will attempt to find an owner for unowned instance attributes.” This text should not be considered to be part of the standard.

Rationale: The first sentence states a requirement that the RTI be responsible for something without explaining the mechanism by which the RTI is expected to fulfill this requirement. In other interpretations in this document (interpretations for

Unconditional Attribute Ownership Divestiture and Resign Federation Execution), text is suggested that explains the mechanism according to which the RTI is expected to fulfill its responsibility for attempting to find an owner for unowned instance attributes when they become unowned as a result of the use of an explicit ownership management service or directive. However, there is no suggestion that the RTI should try to find an owner for unowned instance attributes when they either become unowned as a result of registration or unpublishing a class attribute, or when a non-owning federate that was not previously eligible to own the unowned instance attribute becomes eligible to do so.

The second sentence, about the RID, is simply out of place in the standard, as it refers to implementation-specific behavior. Furthermore, it would allow such a wide variety of RTI behavior that it would make testing infeasible.

1.4.5.2 Figure 15: Establishing Ownership of Instance Attribute (i, k, j)

Interpretation 1

There shall be an additional transition from the *Completing Divestiture* state into the *Waiting for a New Owner to be Found* state that has as a label "[Confirm Divestiture and *NoAcquisitionPending* exception thrown]."

Rationale: This new transition corresponds to interpretation 1 of Service 7.6: Confirm Divestiture.

Interpretation 2

The transition from the *Completing Divestiture* state into the *Able to Acquire* state that currently has as the label "Confirm Divestiture," should have its label modified to read, "Confirm Divestiture (successful)."

Rationale: This new transition corresponds to interpretation 1 of Service 7.6: Confirm Divestiture.

Interpretation 3

There shall be a history state added within the Acquisition Pending state, and there shall be a self-transition from this history state to itself, labeled "*Attribute Ownership Acquisition*."

Rationale: This new transition corresponds to interpretation 1 of Service 7.8: Attribute Ownership Acquisition.

Interpretation 4

There shall be an additional transition added that is a self-transition from the *Willing to Acquire* state to itself that is labeled "*Attribute Ownership Acquisition If Available* [not in '*Acquisition Pending'*]."

Rationale: This new transition corresponds to interpretation 1 of Service 7.9: Attribute Ownership Acquisition If Available.

Interpretation 5

Modify the statechart in Figure 15 so that the Request Attribute Ownership Release transition has a guard added to it that says, “[not in “Waiting for a New Owner to be Found” ^ not in “Completing Divestiture”].”

Rationale: This guard is required to make the statechart consistent with Interpretation 2 of the Attribute Ownership Acquisition service elsewhere in this document.

1.4.5.3 Section 7.1.4: User-supplied tags

Interpretation 1

If an RTI-invoked service is not the result of a federate-invoked service, but the RTI-invoked service has a user-supplied tag as a mandatory argument, the user-supplied tag shall be present in the service invocation, but empty. For example, in the Java API, the tag should be an empty (zero-length) array.

Rationale: This section explains that the user-supplied tags that are present in some federate-invoked services shall be present in the specified resulting RTI-invoked services. It does not explain, however, what the user-supplied tags should be in those RTI-invoked services that are not the result of a federate-invoked service.

For example, according to clause 7.1.4, the user-supplied tag present in the Negotiated Attribute Ownership Divestiture service shall be present in any resulting Request Attribute Ownership Assumption † service invocations. However, the RTI may invoke the Request Attribute Ownership Assumption † service at a federate in an attempt to find an owner for an unowned instance attribute. In this case, the Request Attribute Ownership Assumption † service invocation is not the result of a Negotiated Attribute Ownership Divestiture service invocation by another federate. In this case, it is not clear what the content of the user-supplied tag in the Request Attribute Ownership Assumption † service should be. The user-supplied tag, however, is a mandatory argument of the Request Attribute Ownership Assumption † service. Therefore, if the Request Attribute Ownership Assumption † service is not the result of a previous corresponding Negotiated Attribute Ownership Divestiture service invocation, the user-supplied tag shall be present in the Request Attribute Ownership Assumption † service, but it shall be empty.

Other examples in which this interpretation is relevant are:

- *Attribute Ownership Acquisition Notification †: If an Attribute Ownership Acquisition Notification † service invocation is received at a federate as a result of an owning federate invoking the Unconditional Attribute Ownership Divestiture service, the Attribute Ownership Divestiture If Wanted service, or the Unpublish Object Class Attributes service, or as a result of the owning federate resigning, the user-supplied tag that shall be present in the Attribute Ownership Acquisition Notification † service shall be empty.*
- *Provide Attribute Value Update †: If a Provide Attribute Value Update † service invocation is received at a federate as a result of the Auto-Provide Switch being enabled, the content of the user-supplied tag that shall be present in the Provide Attribute Value Update † service shall be empty.*

- *Request Attribute Ownership Assumption †: If a Request Attribute Ownership Assumption † service invocation is received at a federate as a result of the owning federate invoking the Unconditional Attribute Ownership Divestiture service, the Unpublish Object Class Attributes service, or as a result of the owning federate resigning, the user-supplied tag that shall be present in the Request Attribute Ownership Assumption † service invocation shall be empty.*

1.4.5.4 Service 7.2: Unconditional Attribute Ownership Divestiture

Interpretation 1

For each instance attribute that becomes unowned as a result of invocation of this service, if no joined federates are in either the "Acquiring" or "Willing to Acquire" state with respect to the specified instance attribute, the RTI shall try to identify other joined federates that are willing to own the instance attribute. If any joined federate is in either the "Acquiring" or "Willing to Acquire" state with respect to the specified instance attribute, the RTI may try to identify other joined federates that are willing to own the instance attribute. The mechanism that the RTI shall use to try to identify other joined federates that are willing to own the instance attribute is invocation of the Request Attribute Ownership Assumption † service at all joined federates that are both eligible to own the instance attribute and not in either the "Acquiring" or "Willing to Acquire" state with respect to the specified instance attribute.

Rationale: The Unconditional Attribute Ownership Divestiture service description does not explain or even mention the mechanism by which the RTI is expected to try to find an owner for unowned instance attributes that become unowned as the result of the invocation of the Unconditional Attribute Ownership Divestiture service. The text in this interpretation explains this mechanism. If there are no federates that are trying to acquire an unowned instance attribute, then the RTI must use the Request Attribute Ownership Assumption † service as the mechanism for offering ownership of the unowned instance attribute to federates that are eligible to own it. If there is one or more federate that is trying to acquire an unowned instance attribute, then the RTI may either give ownership of the instance attribute to one of the federates that are trying to acquire it without offering ownership of it to other eligible federates, or it may use the Request Attribute Ownership Assumption † service to offer ownership of the unowned instance attribute to eligible federates before granting ownership of the attribute to a federate that expresses an interest in acquiring it.

1.4.5.5 Service 7.3: Negotiated Attribute Ownership Divestiture

Interpretation 1

The following text should be added to the introductory paragraphs for this service,

> If no joined federates are in either the "Acquiring" or "Willing to Acquire" state with respect to the specified instance attribute, the RTI shall try to identify other joined federates that are willing to own the instance attribute. If any joined federate is in either the "Acquiring" or "Willing to Acquire" state with respect to the specified instance attribute, the RTI may, but is not required to, try to identify other

joined federates that are willing to own the instance attribute. The mechanism that the RTI shall use to try to identify other joined federates that are willing to own the instance attribute is invocation of the Request Attribute Ownership Assumption † service at joined federates that are both eligible to own the instance attribute and not in either the "Acquiring" or "Willing to Acquire" state with respect to the specified instance attribute.

Rationale: This makes clear the mechanism that the RTI is expected to use to try to find federates that are willing to accept ownership of the instance attributes that the owning federate is trying to divest. For each instance attribute that is trying to be divested, if there are no federates that are trying to acquire that instance attribute, then the RTI must use the Request Attribute Ownership Assumption † service as the mechanism for offering ownership of that instance attribute to federates that are eligible to own it. If there is one or more federate that is already trying to acquire that instance attribute, then the RTI may simply invoke the Request Divestiture Confirmation † service at the divesting federate to inform it that a federate that is willing to accept ownership of the instance attribute has been located. It is also acceptable in this situation for the RTI to use the Request Attribute Ownership Assumption † service to offer ownership of that instance attribute to other eligible federates (ones that are not already trying to acquire it) before invoking the Request Divestiture Confirmation † service at the divesting federate to inform it that a federate that is willing to accept ownership of the instance attribute has been located.

Interpretation 2

The sentence in the *Negotiated Attribute Ownership Divestiture* service description that now reads:

"The invoking joined federate shall continue its update responsibility for the specified instance attributes until it divests ownership via the Confirm Divestiture service."

should be changed to read simply:

"The invoking joined federate shall continue its update responsibility for the specified instance attributes until it divests ownership."

Rationale: The invoking joined federate needs to continue its update responsibility for all instance attributes that it owns for as long as it owns them and until it divests them. The manner in which it divests them is irrelevant. By including the words "via the Confirm Divestiture service" at the end of the above sentence, the text as it currently appears in the standard implies that if the federate were to divest ownership by some other means, such as unpublishing, invoking the Unconditional Attribute Ownership Divestiture service, or invoking the Attribute Ownership Divestiture If Wanted service, then the federate would still continue its update responsibility for the instance attributes, even though the federate would no longer be the owner of the instance attributes. This implication is incorrect and contradicts other portions of the specification which prevent a federate from updating an instance attribute if it is not the owner of that instance attribute. Removing the phrase "via the Confirm Divestiture Service" eliminates the incorrect implication.

1.4.5.6 Service 7.4: Request Attribute Ownership Assumption †

Interpretation 1

The following additional pre-condition should be added to this service, “The joined federate is not in either the “Acquiring” or the “Willing to Acquire” state for this instance attribute.”

Rationale: This additional pre-condition is already a requirement as expressed by the guard on the Request Attribute Ownership Assumption transition in the statechart in Figure 15.

1.4.5.7 Service 7.6: Confirm Divestiture

Interpretation 1

There shall be an additional pre-condition to this service that says:

> “There is at least one federate in the federation that is in either the "Acquisition Pending” or the “Willing to Acquire” state with respect to the specified instance attribute.”

There shall be an additional exception to this service that says:

> “There is no joined federate that has an acquisition pending or that is willing to acquire the instance attribute.”

There shall be additional introductory text that says:

> If a federate invokes the Confirm Divestiture service and, as a result, an exception is thrown indicating that there is no joined federate that has an acquisition pending or that is willing to acquire the instance attribute, then that federate shall transition from the *Completing Divestiture* state with regard to that instance attribute into the *Waiting for a New Owner to be Found* state with regard to that instance attribute.

Rationale: If there are no federates in the federation execution that are in the “Acquisition Pending” or “Willing to Acquire” state with respect to the specified instance attribute, the federate that owns the instance attribute shall be prohibited from invoking the Confirm Divestiture service for that instance attribute. If the owning federate were to be allowed to invoke the Confirm Divestiture service under these circumstances, this would result in the instance attribute becoming unowned by all federates, which shall never be allowed to happen as a result of a negotiated divestiture. Therefore, the owning federate shall be prevented from invoking the Confirm Divestiture service under these circumstances.

1.4.5.8 Service 7.7: Attribute Ownership Acquisition Notification †

Interpretation 1

Post-condition (b) of this service incorrectly implies that a federate could invoke the *Attribute Ownership Acquisition* service or the *Attribute Ownership Acquisition If Available* service on an instance attribute that the federate already owns. Post-condition (b) says:

b) The joined federate may stop publishing the corresponding class attributes at the known class of the specified object instance if **it does not own any** corresponding instance attributes for which the joined federate has either:

1. invoked the Attribute Ownership Acquisition service, and has not yet received a corresponding invocation of either the Confirm Attribute Ownership Acquisition Cancellation † service or the Attribute Ownership Acquisition Notification † service, or
2. invoked the Attribute Ownership Acquisition If Available service, and has not yet received a corresponding invocation of either the Attribute Ownership Unavailable † service or the Attribute Ownership Acquisition Notification † service, or
3. invoked the Attribute Ownership Acquisition If Available service and has subsequently invoked the Attribute Ownership Acquisition service [after which condition 1) applies]**.....**

The first sentence of above text should read (changes shown in boldface), “The joined federate may stop publishing the corresponding class attributes at the known class of the specified object instance **if there are no** corresponding instance attributes…”

1.4.5.9 Service 7.8: Attribute Ownership Acquisition

Interpretation 1

If a federate invokes the *Attribute Ownership Acquisition* service for an instance attribute that is already in the“Acquisition Pending” state, that instance attribute's state shall remain unchanged. In other words, if a federate invokes the *Attribute Ownership Acquisition* service for an instance attribute that is in the “Acquiring” state, that instance attribute shall continue to be in the “Acquiring” state and the federate that owns the instance attribute shall not receive a corresponding *Request Attribute Ownership Release †* callback. If there are additional instance attributes in the attribute set that is an argument to the Attribute Ownership Acquisition service, those instance attributes shall enter the “Acquiring” state, assuming that they meet all of the pre-conditions of the *Attribute Ownership Acquisition* service and they are not already in the “Acquiring” or the “Trying to Cancel Acquisition” state, and the federate that owns these instance attributes shall receive a corresponding *Request Attribute Ownership Release †* callback, if appropriate (see the next interpretation for when it is appropriate).

Likewise, if a federate invokes the *Attribute Ownership Acquisition* service for an instance attribute that is in the "Trying to Cancel Acquisition" state, that instance attribute shall continue to be in the "Trying to Cancel Acquisition" state and the federate that owns the instance attribute shall not receive a corresponding *Request Attribute Ownership Release †* callback. If there are additional instance attributes in the attribute set that is an argument to the *Attribute Ownership Acquisition* service, those instance attributes shall enter the "Acquiring" state, assuming that they meet all of the pre-conditions of the *Attribute Ownership Acquisition* service and they are not already in the "Acquiring" or the "Trying to Cancel Acquisition" state, and the federate that owns these instance attributes shall receive a corresponding *Request Attribute Ownership Release †* callback, if appropriate.

Interpretation 2

The introductory text to the *Attribute Ownership Acquisition* service says:

> "If a specified instance attribute is owned by another joined federate, the RTI shall invoke the *Request Attribute Ownership Release* † service for that instance attribute at the owning joined federate."

This text should be replaced with the following:

> "If a specified instance attribute is owned by another joined federate, and that owning federate is in the "Not Divesting" state with respect to the instance attribute, the RTI shall invoke the *Request Attribute Ownership Release †* service for that instance attribute at the owning joined federate. If a specified instance attribute is owned by another joined federate, and that owning federate is in the "Waiting for a New Owner to be Found" state with respect to the instance attribute, the RTI shall not invoke the *Request Attribute Ownership Release †* service for that instance attribute at the owning joined federate, but it shall invoke the *Request Divestiture Confirmation †* service for that instance attribute at the owning joined federate."

Rationale: The text as originally written implies that if an owning federate is in the "Waiting for a New Owner to be Found" state and another federate invokes the Attribute Ownership Acquisition service, the owning federate will receive both a Request Attribute Ownership Release † and a Request Divestiture Confirmation † callback. Furthermore, a potentially large number of eligible federates could invoke the Attribute Ownership Acquisition service. If many federates invoke the Attribute Ownership Acquisition service, the owning federate will receive a corresponding large number of Request Attribute Ownership Release † callbacks while in the "Completing Divestiture" state, and these Request Attribute Ownership Release † callbacks are useless. The expectation that the owning federate would receive both the Request Divestiture Confirmation † callback and numerous useless Request Attribute Ownership Release † callbacks is non-sensical. It requires additional processing by both the RTI and the federate without providing any added value. Furthermore, the standard already prohibits the mirror image of this situation, which involves the question of whether a federate that is already in the "Willing to Acquire" or "Acquisition Pending" state should receive equally useless invocations of the Request Attribute Ownership Assumption † callback. Therefore, in order for the "Owned" state

of ownership management to be consistent with the "Unowned" state of ownership management, and to eliminate unnecessary inefficiency, the text should be changed as described above.

1.4.5.10 Service 7.9: Attribute Ownership Acquisition If Available

Interpretation 1

If a federate invokes the *Attribute Ownership Acquisition If Available* service for an instance attribute that is already in the "Willing to Acquire" state, that instance attribute's state shall remain unchanged. In other words, if a federate invokes the *Attribute Ownership Acquisition If Available* service for an instance attribute that is in the "Willing to Acquire" state, that instance attribute shall continue to be in the "Willing to Acquire" state. If there are additional instance attributes in the attribute set that is an argument to the *Attribute Ownership Acquisition If Available* service, those instance attributes shall enter the "Willing to Acquire" state, assuming that they meet all of the pre-conditions of the *Attribute Ownership Acquisition If Available* service and they are not already in the "Willing To Acquire" state.

Interpretation 2

The introductory text to the *Attribute Ownership Acquisition If Available* service says:

> "For each of the specified instance attributes, the joined federate shall receive either a corresponding *Attribute Ownership Acquisition Notification* † service invocation or a corresponding *Attribute Ownership Unavailable* † service invocation."

This text should be changed to read:

> "For each of the specified instance attributes, the joined federate may receive only a corresponding *Attribute Ownership Acquisition Notification* † service invocation or a corresponding *Attribute Ownership Unavailable* † service invocation."

Rationale: Although it was the case that in the 1.3 (Non-IEEE) version of the HLA Interface Specification the Attribute Ownership Acquisition If Available service was always guaranteed to result in either a corresponding Attribute Ownership Acquisition Notification † callback or a corresponding Attribute Ownership Unavailable † callback, this is no longer the case in the IEEE HLA 1516.1-2000 standard. Because a federate that owns an instance attribute and is in the process of divesting it has the option of staying in the "Completing Divestiture" state indefinitely, it is not the case that a federate that invokes the Attribute Ownership Acquisition If Available service will necessarily receive either an Attribute Ownership Acquisition Notification † service invocation or a corresponding Attribute Ownership Unavailable † service invocation. If the owning federate stays in the "Completing Divestiture" state indefinitely, the federate that invoked the Attribute Ownership Acquisition If Available service will receive neither a corresponding Attribute Ownership Acquisition Notification † service invocation nor a corresponding Attribute Ownership Unavailable † service invocation.

1.4.5.11 Service 7.10: Attribute Ownership Unavailable †

Interpretation 1

Post-condition (b) of this service, like post-condition (b) of the *Attribute Ownership Acquisition Notification* † service (as amended by these Interpretations), should say,

b) The joined federate may stop publishing the corresponding class attributes at the known class of the specified object instance if **there are no** corresponding instance attributes for which the joined federate has either:

1. invoked the Attribute Ownership Acquisition service, and has not yet received a corresponding invocation of either the Confirm Attribute Ownership Acquisition Cancellation † service or the Attribute Ownership Acquisition Notification † service, or

2. invoked the Attribute Ownership Acquisition If Available service, and has not yet received a corresponding invocation of either the Attribute Ownership Unavailable † service or the Attribute Ownership Acquisition Notification † service, or

3. invoked the Attribute Ownership Acquisition If Available service and has subsequently invoked the Attribute Ownership Acquisition service [after which condition 1) applies].

Rationale: When a federate receives the Attribute Ownership Unavailable† callback for an instance attribute, it is no longer in the "Willing to Acquire" state with respect to that instance attribute. Therefore, it is permissible for the federate to stop publishing that instance attribute's corresponding class attribute at the known class of the object instance, providing that there are no other corresponding instance attributes at that same known class that the federate is either "Willing to Acquire," or for which the federate has an "Acquisition Pending." If there is at least one other such instance attribute that the federate is "Willing to Acquire" or for which the federate has an "Acquisition Pending," then the federate must be prohibited from unpublishing the corresponding class attribute of that instance attribute at the known class of the object instance. Stipulations 1, 2, and 3 above ensure that there are no other such instance attributes that the federate is "Willing to Acquire" or for which the federate has an "Acquisition Pending."

1.4.5.12 Service 7.15: Confirm Attribute Ownership Acquisition Cancellation †

Interpretation 1

Post-condition (b) of this service, like post-condition (b) of both the *Attribute Ownership Acquisition Notification †* service and the *Attribute Ownership Unavailable †* service (both as amended by these Interpretations), should say,

b) The joined federate may stop publishing the corresponding class attributes at the known class of the specified object instance if **there are no** corresponding instance attributes for which the joined federate has either:

1. invoked the Attribute Ownership Acquisition service, and has not yet received a corresponding invocation of either the Confirm Attribute Ownership Acquisition Cancellation † service or the Attribute Ownership Acquisition Notification † service, or
2. invoked the Attribute Ownership Acquisition If Available service, and has not yet received a corresponding invocation of either the Attribute Ownership Unavailable † service or the Attribute Ownership Acquisition Notification † service, or
3. invoked the Attribute Ownership Acquisition If Available service and has subsequently invoked the Attribute Ownership Acquisition service [after which condition 1) applies].

Rationale: The rationale for this interpretation is the same as for service 7.10 above.

1.4.6 Time Management Interpretations

1.4.6.1 Figure 16: Temporal State statechart

Interpretation 1

Two of the service names on the transition from the “Idle” to the “Time Advancing” state are not correct. “Next Event Request” should instead be “Next Message Request” and “Next Event Request Available” should be “Next Message Request Available.” The label from the Asynchronous Delivery Enabled state to the Asynchronous Delivery Disabled state has a typographical error. It should be “Disable Asynchronous Delivery” instead of “Disable Asynchronously Delivery.”

1.4.6.2 Service 8.12 Flush Queue Request

Interpretation 1

Lines 4-8 and lines 9-11 of the introductory text to this service conflict with each other:

Lines 4-8 now read,

The RTI shall advance the joined federate’s logical time to the smallest of the following:

- the specified logical time
- the joined federate's GALT value
- the smallest time stamp of all TSO messages delivered by the RTI in response to this invocation of the Flush Queue Request service.

Lines 9-11 now read:

If the joined federate will not receive any additional TSO messages with time stamps less than the specified logical time, the joined federate shall be advanced to the specified logical time. Otherwise, the RTI shall advance the joined federate's logical time as far as possible (i.e., to the joined federate's GALT).

Lines 9-11 should be deleted, so that the above text now reads simply,

The RTI shall advance the joined federate's logical time to the smallest of the following:

- the specified logical time
- the joined federate's GALT value
- the smallest time stamp of all TSO messages delivered by the RTI in response to this invocation of the Flush Queue Request service.

1.4.7 Data Distribution Management Interpretations

1.4.7.1 Section 9.1: DDM Overview

Interpretation 1

According to clause 9.1.1 (c), "A region specification shall be a set of ranges." Whereas according to clause 9.1.3.1 (a), "A region specification may be created using the *Create Region* service." These two statements are contradictory, because the *Create Region* service only determines what dimensions will be in a region, not the ranges of those dimensions. A region specification is a set of ranges. The only way that a region specification can be created is by a federate successfully invoking the following services, in order:

1. Invoke the *Create Region* service (DDM Service 9.2) to create a region with a specific set of dimensions;
2. Invoke the *Set Range Bounds* service (Support Service 10.32) for every dimension that was explicitly specified when that region was created, to set the lower and upper bounds of the range of that dimension for that region;
3. Invoke *Commit Region Modifications* (DDM Service 9.3) to inform the RTI about the changes to the ranges of the dimensions specified in the preceding series of *Set Range Bounds* service invocations.

A region *template* is created when a federate invokes the *Create Region* service. The region designator argument that is returned as a result of the *Create Region* service is a designator of a region template only. It is not the designator of a region specification, because the range bounds have not yet been set for all of the dimensions of the region template followed by a successful invocation of the *Commit Region Modifications* service for this region. Only after the *Set Range Bounds* service has been successfully invoked for every dimension in the region, followed by a successful invocation of the *Commit Region Modifications* service, is that region designator the designator of a region specification.

When a federate invokes the *Register Object Instance With Region*, *Associate Regions For Updates*, *Subscribe Object Class Attributes With Regions*, *Subscribe Interaction Class With Regions*, *Send Interaction With Regions*, or *Request Attribute Value Update With Regions* service with that region specification designator as an argument, one or more region realizations is created. These region realizations, however, do not have designators.

Interpretation 2

Contrary to what is said in clause 9.1.3.1 (b), modifying a region does not determine the specified dimensions of a region realization. Item (b) should have "or modified" deleted from its end. That is, the specified dimensions of the region shall be the dimensions that are explicitly provided when the *Create Region* service is invoked to create that region. Invocation of neither the *Set Range Bounds* service nor the *Commit Region Modifications* service for a particular region has any effect on what the specified dimensions of that region are.

Interpretation 3

Contrary to what is said in clause 9.1.3.1 (d), the *Commit Region Modifications* service cannot be used to create a region realization from a region specification. The *Commit Region Modifications* service can only either:

- create a region specification from a region template, or
- modify the range bounds of an existing region specification and thereby also modify the range bounds of all existing region realizations that are derived from that region specification.

Interpretation 4

If a federate invokes the *Create Region* service but does not subsequently successfully invoke the *Set Range Bounds* service for every dimension in the region, followed by a successful invocation of the *Commit Region Modifications* service for the region, then the federate has not created a region specification. In particular:

- If a federate invokes the *Create Region* service, followed by an invocation of the *Set Range Bounds* service for none or some, but not for all, of the dimensions that were specified when the region was created, followed by an invocation of the *Commit Region Modifications* service for that region, the *Commit Region Modifications* service shall throw the "Invalid region" exception, because each region designator that is passed to the *Commit Region Modifications* service is required to have had the *Set Range Bounds* service invoked at least once for all of its dimensions. The effects of the *Set Range Bounds* service invocations that were made, if any, are still pending. They will take affect if and when the *Set Range Bounds* service has been invoked at least once for all of the dimensions of the region, followed by the invocation of the *Commit Region Modifications* service for that region.
- If a federate invokes the *Create Region* service, followed by at least one *Set Range Bounds* service invocation for every one of the dimensions that were specified when the region was created, but does not invoke the *Commit Region Modifications* service, the region continues to be only a region template, but not a region

specification. The effects of the *Set Range Bounds* service invocations are still pending and will take effect if and when the *Commit Region Modifications* service is successfully invoked for that region.

The effects of invocation of the *Set Range Bounds* service for a given region (template or specification) will remain pending until the *Commit Region Modifications* service is subsequently invoked for that region. If a federate invokes the *Set Range Bounds* service repeatedly for a given dimension of a given region before invoking the *Commit Region Modifications* service for that region, the range values provided in the most recent invocation of the *Set Range Bounds* service will become the range values for that dimension of that region specification.

Rationale: If the Set Range Bounds service is not called for a given dimension of a region, or the Commit Region Modifications service is not called after the Set Range Bounds service has been invoked for every dimension of that region, then the region will contain one or more dimensions that do not have ranges set, and there will be no way to use this region meaningfully. If some of a region's dimensions do not have ranges, then there is no way to determine whether or not the region overlaps with other regions. In summary, although the creation of a region specification is accomplished using a sequence of service calls, all of these service calls must be invoked successfully, in sequence, in order to successfully create a region specification.

1.4.7.2 Section 9.1.2: Default Ranges

Interpretation 1

According to 9.1.2 (d), “Each dimension in the FDD shall have either a default range specified in terms of [0, the dimension's upper bound) or shall have…” To clarify, the default range shall be specified in terms of *the bounds* [0, the dimension's upper bound); it is not necessary that the default range cover the entire dimension.

1.4.7.3 DDM: "Invalid Region" exceptions

Interpretation 1

Some DDM services require that the region designator used as an argument be the designator of a region specification, whereas other DDM services require that the region designator used as an argument be the designator of either a region template or a region specification. One DDM service, *Commit Region Modifications*, requires that the region designator used as an argument be that of either a region template that has had the *Set Range Bounds* service called at least once for all of its dimensions, or a region specification. If a DDM or Support service is invoked with a region designator argument that is not of the variety it is expecting, the service shall throw the "Invalid region" exception. Hence, the circumstances that shall cause the "Invalid region" exception to be thrown will vary from service to service, depending on the type of region entity (template, specification, or template with all range bounds set) for which the service is expected to receive a designator.

For all services that can be used to create a region realization from a region specification, the region designator argument shall be the designator of a region specification. Specifically, the following services, which result in the creation of region realizations from region specifications, require that the region designator argument be the designator of a region specification: *Register Object Instance With Regions*, *Associate Regions For Updates*, *Subscribe Object Class Attributes With Regions*, *Subscribe Interaction Class With Regions*, *Send Interaction With Regions*, and *Request Attribute Value Update With Regions*.

The following services also require that the region designator argument be the designator of a region specification: *Unassociate Regions for Updates*, *Unsubscribe Object Class Attributes With Regions*, and *Unsubscribe Interaction Class With Regions*.

The region designator argument for the *Delete Region* service shall be the designator of either a region template or a region specification.

There are also three Support Services that take a region designator as argument:

- The region designator argument for the *Get Range Bounds* service shall be the designator of either a region specification or a region realization.
- The region designator argument for the *Get Dimension Handle Set* service shall be the designator of a region template, a region specification, or a region realization.
- The region designator argument for the *Set Range Bounds* service shall be the designator of either a region template or a region specification.

1.4.7.4 Section 9.1.7: Convey Region Designator Sets Switch

Interpretation 1

According to the fourth paragraph of section 9.1.7, whenever the Convey Region Designator Sets Switch is enabled, “the optional Set of Sent Region Designators argument shall be provided (as appropriate) with all *Reflect Attribute Values †* and *Receive Interaction †* service invocations at all joined federates.”

The words “as appropriate” should be understood to mean that the Set of Sent Region Designators argument shall be provided in a Reflect Attribute Values service invocation or in a Receive Interaction service invocation if and only if:

- the Convey Region Designator Sets Switch is enabled, and
- the reflected instance attribute(s) or the received interaction has available dimensions.

This means that if the above conditions are true, all federates, not just those federates that are using DDM or that are subscribed to the specified attributes or interaction class with regions, will receive the reflect or interaction with a set of sent region designators argument.

Rationale: In Clause 6.7 of IEEE 1516.1-2000, it is made clear that if the specified instance attributes have available dimensions and the Convey Region Designator Sets Switch is enabled, the set of sent region designators argument of the Reflect Attribute

Values † service shall contain the update region set, if any, that was used for update of the instance attributes at the joined federate which invoked the corresponding Update Attribute Values service. In clause 6.9 it is made clear that if the specified interaction has available dimensions and the Convey Region Designator Sets Switch is enabled, the set of sent region designators argument of the Receive Interaction † service shall contain the update region set, if any, that was supplied to the corresponding Send Interaction With Regions service invocation by the sending joined federate. The Convey Region Designator Sets Switch is a federation-wide switch, and therefore affects all federates in the federation, regardless of whether or not they are using DDM.

Interpretation 2

According to clause 6.1.4, when the Convey Region Designator Sets Switch is enabled and a *Reflect Attribute Values* † callback is received, all of the instance attribute/value pairs in that *Reflect Attribute Values* † service invocation must have the same set of sent region designators. The set of sent region designators conveyed is not a set of designators of region specifications or a set of designators of region realizations. It is, instead, a set of designators of **copies of the update region realizations**. The range values of each of the dimensions in each of these region realization copies are the same as the range values of each of the dimensions of the corresponding update region realization that were in effect when the region overlap calculation was performed. Each designator is guaranteed to refer to a particular region realization copy, and the copy is guaranteed to remain in tact, until the *Reflect Attribute Values* † service or the *Receive Interaction* † service invocation at the receiving/reflecting federate completes. No change to the range values of the region realization from which the copy was made will cause a change to the range values of the copy as long as the *Reflect Attribute Values* † service or the *Receive Interaction* † service invocation is still in progress.

Consider the following example:

Assume the following FDD information:

Dimension d1 [0, 1000) default range [1, 100)
Dimension d2 [0, 1000) default range [1, 100)
Dimension d3 [0, 1000) default range [1, 100)
Dimension d4 [0, 1000) default range [1, 100)

Object Class C
Attribute X dimensions d1, d3
Attribute Y dimensions d1, d4
Attribute Z dimensions d2

Now assume the following service invocations:

Create Region ({d1}) return (designator R)
Create Region ({d2}) return (designator G)

At this point, R and G are the designators of region templates only.

Set Range Bounds(R, d1, 2, 45)
Commit Region Modifications (R)
Set Range Bounds (G, d2, 5, 15)
Commit Region Modifications (G)

At this point, R and G are the designators of region specifications.

Register Object Instance(object class C, “object1”)
Associate Regions For Updates (object1, ({X, Y}, {R}))

After the above *Associate Regions For Updates* (object1, ({X, Y}, {R})) service invocation, R is still the designator of a region specification; however, as a result of the *Associate Regions For Updates* (object1, ({X, Y}, {R})) service invocation, two region realizations were created: one associated with X, and one associated with Y. The federate that created the region specification R, however, has no designator that refers to either of these region realizations uniquely. The region realization that is associated with X has specified dimension d1 and unspecified dimension d3. The region realization that is associated with Y has specified dimension d1 and unspecified dimension d4.

Associate Regions For Updates (object1, ({Z}, {G}))

Use a MOM interaction to enable the Convey Region Designator Sets Switch.

Update Attribute Values (object1, {(X, 44), (Y, 53), (Z, 66)}, “user tag”)

At this point, all federates that are subscribed to X, Y and Z at the appropriate object class will receive three separate reflects:

- one that includes only an attribute/value pair for X,
- one that includes only an attribute/value pair for Y, and
- another that includes only an attribute/value pair for Z.

Note that although the region realization associated with X and the region realization associated with Y are derived from the same region specification (region specification R), the region realization associated with X differs from the region realization associated with Y because the range values of the unspecified dimensions of each of the region realizations differ. This passelization of a single update into three different reflects is required by the fact that different region realizations were associated with X,Y, and Z. The reflect for X will contain a Set of Sent Region Designators consisting of one designator, R1, which has range values for dimensions d1 (2, 45) and d3 (1,100); the reflect for Y will contain a Set of Sent Region Designators consisting of one designator, R11, which has range values for dimensions d1 (2, 45) and d4 (1,100); and the reflect for Z will contain a Set of Sent Region Designators consisting of one designator, G1, which has range values for only dimension d2 (5, 15).

If the reflecting federate were to invoke *Get Dimension Handle Set* (R1) while the Reflect Attribute Values service were still in progress and before it is allowed to complete, it would get back a response of {d1, d3}, because R1 is a copy of a region realization consisting of specified dimension d1 and unspecified dimension d3. If the reflecting federate were to invoke *Get Range Bounds* (R1, d1), it would get return values of: lower bound 2; upper bound 45. If the reflecting federate were

to invoke *Get Range Bounds* (R1, d3), it would get return values of: lower bound 1; upper bound 100. However, once the *Reflect Attribute Values* service invocation completes at the reflecting federate, the reflecting federate is no longer guaranteed that the values returned by *Get Range Bounds* (R1, d1) will still be 2 and 45.

For the second part of this example, suppose that the updating federate were to invoke the following:

Update Attribute Values (object1, {(X, 100)}, “user tag”).

The reflecting federate would receive a reflect with an attribute/value pair for (X, 100) in it, and a conveyed region set consisting of one designator, R2. R2 may or may not be equal to R1. That is an implementation detail. If the reflecting federate were to invoke *Get Range Bounds* (R2, d1) before the Reflect Attribute Values service invocation were allowed to complete, it would get return values of: lower bound 2; upper bound 45.

Suppose the updating federate were to invoke *Set Range Bounds*(R, d1, 10, 20), followed by *Commit Region Modifications* (R) while the previous *Reflect Attribute Values* service invocation is still in progress at the reflecting federate.

If the reflecting federate were to invoke *Get Range Bounds* (R2, d1), it would still get return values of: lower bound 2; upper bound 45, because the Reflect Attribute Values service invocation that contains the region realization copy with these range values has not yet completed.

Suppose the updating federate were to invoke *Update Attribute Values* (object1, {(X, 200)}, “user tag”) after the previous *Reflect Attribute Values* service invocation had completed. The reflecting federate would receive a reflect with an attribute/value pair for (X, 200) in it, and a conveyed region set consisting of one designator, R3. (R3 may or may not be equal to R1 or R2; that is an implementation detail.)

If the reflecting federate were to invoke *Get Range Bounds* (R3, d1) while the reflect is in progress, it would now get new return values of: lower bound 10; upper bound 20, because the updating federate had modified the corresponding region specification before invoking the *Update Attribute Values* service that resulted in the current reflect.

Rationale: The conveyed region sets must contain designators of copies of region realizations, rather than designators of region realizations themselves, because if the designators of the actual region realizations were to be conveyed, this would result in an undesirable race condition. If the designators were of actual region realizations, for example, rather than of just copies of region realizations, then after a federate invokes an Update Attribute Values service or a Send Interaction with Regions service that results in a set of sent region designators being conveyed in the corresponding Reflect Attribute Values † service or Receive Interaction † service, there is a race that could occur between the federate that sent the update or interaction setting new range bounds and committing region modifications to a sent region, and the federate that received the reflect or interaction invoking the Get Dimension Handle Set and Get Range Bounds services to query all of the range bounds of the region realizations received.

According to post-condition (b) of the Commit Region Modifications service, when a region specification is modified, all update region realizations that are derived from that region specification are also modified. Therefore, considering the second part of the example above, suppose the sending federate updates the instance attributes and the reflecting federate receives region specification designator R2 in the Set of Sent Region Designators argument of the reflect, as discussed in the example. Then suppose that the updating federate modifies the region specification R. This means that all of the update region realizations that were derived from R (R2) also get modified. If the updating federate modifies R before the reflecting federate has a chance to use the Get Range Bounds service to determine what the range values of each dimension of R2 are, then the reflecting federate will never be able to determine what the range values of R2 were when the overlap calculation that resulted in the reflect with R2 in it was performed. A race condition would exist between the updating federate's attempt to modify the region specification and the reflecting federate's attempt to determine the range values of each dimension of the derived region realizations received.

Conveying designators of copies of these region realizations, instead of designators of region realizations themselves, eliminates this race condition. The reflecting federate is always guaranteed that if it queries the range bounds using a region designator conveyed in a reflect or received interaction while the reflect or receive interaction is still in progress, the range values of the region specification copy will be the values of the update region realization that were in effect at the time that the overlap calculation was done on the update region set and subscription region set that resulted in that reflect or received interaction.

Interpretation 3

Section 9.1.7, last sentence: This sentence should say, “A joined federate shall use **the region realization designators received in** conveyed region sets only in the *Get Range Bounds* and *Get Dimension Handle Set* service invocations.”

1.4.7.5 Service 9.3 Commit Region Modifications

Interpretation 1

Each region designator that is passed to the *Commit Region Modifications* service is required to be the designator of either a region template that has had the *Set Range Bounds* service invoked at least once for all of its dimensions, or a region specification. If a federate invokes the *Commit Region Modifications* service with a region designator argument that is neither the designator of a region template that has had the *Set Range Bounds* service invoked at least once for all of its dimensions, nor the designator of a region specification, then the *Commit Region Modifications* service shall throw the “Invalid region” exception.

Rationale: The purpose of the Commit Region Modifications service is to either create a region specification from a region template or modify an already-existing region specification and all derived region realizations. If not all dimensions of a region template have had their range bounds set, then no region specification can be created by the Commit Region Modifications service because a region specification, by

definition, has range bounds set for each dimension in the region template. Therefore, invoking the Commit Region Modifications service in this situation is considered an error.

1.4.7.6 Service 9.4: Delete Region

Interpretation 1

The second sentence of this service description should say, "A region in use for subscription or update **shall** not be deleted." If a federate invokes the *Delete Region* service using as an argument the designator of a region that is in use for subscription or update, the *Delete Region* service shall not complete successfully and the "Region is in use for update or subscription" exception shall be thrown at that federate.

Rationale: Deletion of a region that is in use must not be carried out by the RTI, because the expected behavior of the RTI in such a situation is not well-defined. Pre-condition e, "The region is not in use for update or subscription," and exception c, "The region is in use for update or subscription," support the interpretation that deletions of regions that are in use shall not be allowed.

1.4.7.7 Service 9.5: Register Object Instance With Regions

Interpretation 1

The last paragraph of this service description says,

> If the optional object instance name argument is supplied, that name shall have been successfully reserved as indicated by the Object Instance Name Reserved † service and shall be coadunated with the object instance. If the optional object instance name argument is not supplied, the RTI shall create one when needed (Get Object Instance Name service).

With regard to the behavior described in this paragraph, the *Register Object Instance With Regions* service shall behave identically to the *Register Object Instance* service, 6.4. That is, if the optional object instance name argument is not supplied when the *Register Object Instance With Regions* service is invoked, the RTI shall create a federation execution-wide unique name and that name shall be coadunated with the object instance.

Rationale: The registration of all object instances should be treated consistently, whether they are registered with or without regions. All object instances should have federation execution-wide unique names.

Interpretation 2

There shall be a precondition of this service that says, "All supplied region designators are designators of region specifications." If a region designator specified is not a designator of a region specification, the "Invalid region" exception shall be generated.

Rationale: This service shall result in the creation of one or more region realizations. A region realization can only be derived from a region specification, not a region template, because all range bounds of the specified regions of the region realization must be set. If the designator of a region template, instead of a designator of a region specification, is used as an argument to this service, then this service could not result in the creation of a region realization because the range bounds of one or more of the dimensions in the region would not be set.

1.4.7.8 Service 9.6: Associate Regions For Updates

Interpretation 1

This service description says that, "This service shall add the specified regions to the set of associations of each specified instance attribute."

If an instance attribute is associated with the default region, then invocation of the *Associate Regions For Updates* service shall remove the association of that instance attribute with the default region as well as add the association of that instance attribute to the specified region. This is the sole exception to the additive semantics rule.

Rationale: See 9.1.3.2 (a). If an instance attribute is associated with the default region, there is no value in also associating it with other regions, because the default region always overlaps with all other regions that have dimensions. An instance attribute should only be associated with the default region if it is not associated with any other region. When an instance attribute is associated with another region, its association with the default region should no longer exist. So, if an instance attribute is associated with the default region, associating that instance attribute with one or more other regions shall have the effect of removing the association of that instance attribute with the default region, because there is no way to explicitly remove the association of that instance attribute with the default region.

Interpretation 2

There shall be a precondition of this service that says, "All supplied region designators are designators of region specifications." If a region designator specified is not a designator of a region specification, the "Invalid region" exception shall be generated.

Rationale: This service shall result in the creation of one or more region realizations. A region realization can only be derived from a region specification, not a region template, because all range bounds of the specified regions of the region realization must be set. If the designator of a region template, instead of a designator of a region specification, is used as an argument to this service, then this service could not result in the creation of a region realization because the range bounds of one or more of the dimensions in the region would not be set.

1.4.7.9 Service 9.7: Unassociate Regions For Updates

Interpretation 1

The second sentence in this service description says that, “No changes shall be made to the association set if the specified regions are not in the set of associations of the specified instance attributes.” If one or more of the specified regions is in the set of associations, then these regions shall be unassociated from the specified instance attributes; if any of the specified regions are not in the set of associations of the specified instance attributes, they shall be ignored.

Interpretation 2

There shall be a precondition of this service that says, “All supplied region designators are designators of region specifications.” If a region designator specified is not a designator of a region specification, the “Invalid region” exception shall be generated.

Rationale: Only region specifications, not region templates, can be used as arguments to the Associate Regions For Updates service, so it would make no sense to use a region template as argument to the Unassociate Regions For Updates service. A region template cannot be associated for updates, so it therefore cannot be unassociated for updates either.

1.4.7.10 Service 9.8: Subscribe Object Class Attributes With Regions

Interpretation 1

This service description should be clarified regarding the intended use of the optional passive subscription indicator. The text does not explain what it means for a subscription with region to be passive or active. That is, it is not clear whether the active/passive characteristic applies on a per-(object class, class attribute, region) triple basis, on a per-(object class, class attribute) pair basis, or on some other basis. The use of the optional passive subscription indicator is expected to work on the triple basis, which is as follows:

> Each subscribed attribute of a class with region is subscribed either actively or passively (but not both actively and passively) at that given object class and with that particular region. Two different class attributes that are subscribed with regions at the same object class and with the same region may be subscribed differently from each other: one active and one passive, and a class attribute that is subscribed with regions at a given object class but with more than one region may be subscribed differently (either actively or passively) with each region. That is, the active/passive characteristic is a property of a subscription to a class attribute at a given object class with a given region.
>
> Each (object class, class attribute, region) triple specified in a given invocation of the *Subscribe Object Class Attributes With Regions* service will take on the effect of the optional passive/active subscription indicator supplied (or not supplied) with that service invocation. Furthermore, if there is an existing (object class, class attribute, region) subscription that has the same object class, class attribute, and

region value as those specified in the current invocation of the *Subscribe Object Class Attributes With Regions* service, this existing subscription will take on the effect of the optional active/passive subscription indicator supplied (or not supplied) with the current service invocation.

Invoking the *Subscribe Object Class Attributes With Regions* service with an (object class, class attribute, region set) triple such that the region set is empty shall not change the active/passive subscription nature of any of the (object class, class attribute, region) triples that are already subscribed. Each use of the *Subscribe Object Class Attributes With Regions* service shall add the specified regions to the set of subscriptions of the specified class attributes at that object class, if they are not already in this set; and may change the active/passive nature of existing subscriptions if they are.

Rationale: The intent is for invocations of the Subscribe Object Class Attributes With Regions service for any given object class, class attribute, and region to be cumulative with respect to the set of subscribed regions of a given class attribute, but substitutive with respect to whether each (object class, class attribute, region) subscription is subscribed actively or passively. If the current invocation of the Subscribe Object Class Attributes With Regions service includes an (object class, class attribute, region) subscription that already exists, the property of active versus passive for that (object class, class attribute, region) subscription will be substituted according to the value (or absence) of the optional passive subscription indicator argument to the current invocation of the Subscribe Object Class Attributes With Regions service.

Interpretation 2

There shall be a precondition of this service that says, “All supplied region designators are designators of region specifications.” If a region designator specified is not a designator of a region specification, the “Invalid region” exception shall be generated.

Rationale: This service shall result in the creation of one or more region realizations. A region realization can only be derived from a region specification, not a region template, because all range bounds of the specified regions of the region realization must be set. If the designator of a region template, instead of a designator of a region specification, is used as an argument to this service, then this service could not result in the creation of a region realization because the range bounds of one or more of the dimensions in the region would not be set.

1.4.7.11 Service 9.9: Unsubscribe Object Class Attributes With Regions

Interpretation 1

This service begins by saying that, “The *Unsubscribe Object Class Attributes With Regions* service shall inform the RTI that it shall stop notifying the joined federate of object instance discoveries and attribute updates for instance attributes of the specified object class in the specified region.”

This sentence should be understood to mean:

The Unsubscribe Object Class Attributes With Regions service shall require the RTI to remove the specified region from the subscription region set of the specified class attribute at the specified object class, which is used to determine when the Discover Object Instance † service and the Reflect Attribute Values † service shall be invoked at this joined federate.

Rationale: The purpose of the Unsubscribe Object Class Attributes With Regions service is to remove subscriptions with regions such that

- *Those subscriptions will no longer be part of the calculation regarding whether or not the subscription region set for the class attribute at the candidate discovery class at the subscribing joined federate overlaps the update region set of the instance attribute at the owning federate (for purposes of determining whether the subscribing federate should discover the object instance), and*
- *Those subscriptions will no longer be part of the calculation regarding whether or not the subscription region set for the class attribute at the known class of the object instance at the subscribing joined federate overlaps the update region set of the instance attribute at the owning federate at the time of update (for purposes of determining whether the subscribing federate should receive a Reflect Attribute Values † callback when the instance attribute is updated).*

Here is an example: Suppose that federate 1 has a given object class and class attribute subscribed with two different regions, R1 and R2. Suppose that federate 2 owns a corresponding instance attribute of an object instance that it has registered at the given class and that federate 2 has associated region R3 with that instance attribute for updates. Suppose also that region R3 overlaps region R1 and region R3 also overlaps region R2. If federate 1 invokes the Unsubscribe Object Class Attributes With Regions service for that object class, class attribute, and region R1, then federate 1 would still expect to reflect attribute updates for the instance attribute, because it is still subscribed to the corresponding class attribute with region R2. If federate 1 also invokes the Unsubscribe Object Class Attributes With Regions service for that object class, class attribute, and region R2, then federate 1 would not expect to reflect attribute value updates for the instance attribute.

Interpretation 2

There shall be a precondition of this service that says, “All supplied region designators are designators of region specifications.” If a region designator specified is not a designator of a region specification, the “Invalid region” exception shall be generated.

Rationale: Only region specifications, not region templates, can be used as arguments to the Subscribe Object Class Attributes With Regions service, so it would make no sense to use a region template as argument to the Unsubscribe Object Class Attributes With Regions service. A region template cannot be subscribed, so it therefore cannot be unsubscribed either.

1.4.7.12 Service 9.10: Subscribe Interaction Class With Regions

Interpretation 1

The use of the optional passive subscription indicator is expected to work as follows:

Each subscribed interaction class with regions is subscribed either actively or passively with a given region, but not both. Two different interaction classes that are subscribed at the same region may be subscribed differently from each other: one active and one passive, and the same interaction class that is subscribed with two different regions may be subscribed differently (either actively or passively) with each region. Each (interaction class, region) pair specified in a given invocation of the *Subscribe Interaction Class With Regions* service will take on the effect of the optional active/passive subscription indicator supplied (or not supplied) with that service invocation. Furthermore, if there is an existing (interaction class, region) subscription that has the same interaction class and region values as those specified in the current invocation of the *Subscribe Interaction Class With Regions* service, it will take on the effect of the optional passive/active subscription indicator supplied (or not supplied) with the service invocation.

Invoking the *Subscribe Interaction Class With Regions* service with an (interaction class, region set) pair such that the region set is empty shall not change the active/passive subscription nature of any of the (interaction class, region) pairs that are already subscribed. Each use of the *Subscribe Interaction Class With Regions* service shall add the specified regions to the set of subscriptions of the specified interaction class, if they are not already in this set; and may change the active/passive nature of existing subscriptions if they are.

Rationale: The intent is for invocations of the Subscribe Interaction Class With Regions service for any given interaction class to be cumulative with respect to the set of subscribed regions of a given interaction class, but substitutive with respect to whether each (interaction class, region) pair is subscribed actively or passively. If the current invocation of the Subscribe Interaction Class With Regions service includes a given (interaction class, region) subscription that already exists, the property of active versus passive for that (interaction class, region) subscription is substituted according to the value (or absence) of the optional passive subscription indicator argument to the current invocation of the Subscribe Interaction Class With Regions service.

Interpretation 2

There shall be a precondition of this service that says, “All supplied region designators are designators of region specifications.” If a region designator specified is not a designator of a region specification, the “Invalid region” exception shall be generated.

Rationale: This service shall result in the creation of one or more region realizations. A region realization can only be derived from a region specification, not a region template, because all range bounds of the specified regions of the region realization must be set. If the designator of a region template, instead of a designator of a region specification, is used as an argument to this service, then this service could not result in the creation of a region realization because the range bounds of one or more of the dimensions in the region would not be set.

1.4.7.13 Service 9.11: Unsubscribe Interaction Class With Regions

Interpretation 1

This service begins by saying that, "The *Unsubscribe Interaction Class With Regions* service shall inform the RTI that it shall no longer notify the joined federate of interactions of the specified class that are sent into the specified region." This sentence should be understood to mean,

> The Unsubscribe Interaction Class With Regions service shall require the RTI to remove the specified region from the subscription region set of the specified interaction class, which is used to determine when the Receive Interaction † service shall be invoked at this joined federate.

Rationale: The purpose of the Unsubscribe Interaction Class With Regions service is to remove subscriptions with regions for a given interaction class such that those subscriptions with regions will no longer be part of the calculation regarding whether or not the update region set of a sent interaction overlaps the subscription region set for that interaction class.

Here is an example: Suppose that federate 1 has a given interaction class subscribed with two different regions, R1 and R2. Suppose that federate 2 sends an interaction of the given class with region R3. Suppose also that region R3 overlaps region R1 and region R3 also overlaps region R2. If federate1 invokes the Unsubscribe Interaction Class With Regions service for that object class and region R1, federate 1 would still expect to receive an interaction of that class that is sent with region R3 because federate 1 is still subscribed to the interaction class with region R2. If federate 1 also invokes the Unsubscribe Interaction Class With Regions service for that object class and region R2, however, federate 1 would not expect to receive an interaction of that class that is sent with region R3.

Interpretation 2

The second paragraph of the 9.11 service description contains an error. The first sentence reads, "If the region set provided is empty, no subscription to the interaction class shall not be removed." The "not" should not be present in this sentence. It should read, "If the region set provided is empty, no subscriptions to the interaction class shall be removed."

Interpretation 3

There shall be a precondition of this service that says, "All supplied region designators are designators of region specifications." If a region designator specified is not a designator of a region specification, the "Invalid region" exception shall be generated.

Rationale: Only region specifications, not region templates, can be used as arguments to the Subscribe Interaction Class With Regions service, so it would make no sense to use a region template as argument to the Unsubscribe Interaction Class With Regions service. A region template cannot be subscribed, so it therefore cannot be unsubscribed either.

1.4.7.14 Service 9.12: Send Interaction With Regions

Interpretation 1

Clarification: The second sentence of this service description reads, “The interaction parameters shall only be those in the specified class and all super-classes, as defined in the FDD.” Only parameters that are available at that interaction class may be sent in a given interaction, but a federate is not required to send all available parameters of the interaction class with the interaction.

Interpretation 2

There shall be a precondition of this service that says, “All supplied region designators are designators of region specifications.” If a region designator specified is not a designator of a region specification, the “Invalid region” exception shall be generated.

Rationale: This service shall result in the creation of one or more region realizations. A region realization can only be derived from a region specification, not a region template, because all range bounds of the specified regions of the region realization must be set. If the designator of a region template, instead of a designator of a region specification, is used as an argument to this service, then this service could not result in the creation of a region realization because the range bounds of one or more of the dimensions in the region would not be set.

1.4.7.15 Service 9.13: Request Attribute Value Update With Regions

Interpretation 1

There shall be a precondition of this service that says, “All supplied region designators are designators of region specifications.” If a region designator specified is not a designator of a region specification, the “Invalid region” exception shall be generated.

Rationale: This service shall result in the creation of one or more region realizations. A region realization can only be derived from a region specification, not a region template, because all range bounds of the specified regions of the region realization must be set. If the designator of a region template, instead of a designator of a region specification, is used as an argument to this service, then this service could not result in the creation of a region realization because the range bounds of one or more of the dimensions in the region would not be set.

1.4.7.16 DDM Typos

Interpretation 1

Section 9.1.6, second paragraph: the term “DM” should be replaced by “Object Management” in the sentence, “A joined federate using data distribution management services shall interpret all uses of the following three declaration management services by any joined federate in the federation execution (including itself)…”

Interpretation 2

Service 9.5: Register Object Instance With Regions: The returned argument should be an object instance “handle” (not an object instance “designator”).

1.4.8 Support Services Interpretations

1.4.8.1 Section 10.1.2: Advisory Switches

Interpretation 1

This clause says that, “The enabling of an advisory switch directs that the RTI shall inform the joined federate, via the appropriate advisory notifications, whenever the conditions covered by that advisory change.” For purposes of determining whether the conditions covered by each advisory have changed, when the conditions required for sending each advisory become relevant at a given federate, the initial state of each advisory, as known by that federate, is as follows:

1. Before a federate begins publishing an object class, the conditions covered by the Object Class Relevance Advisory switch are assumed to be such that the registration of new object instances of the specified object class is not advised.
2. Before a federate begins publishing an interaction class, the interaction class is in the Interactions Turned Off state with respect to that federate. (see Figure 12)
3. Before a federate becomes the owner of an instance attribute, the instance attribute is in the Updates Turned Off state with regard to that federate. (see Figure 14)
4. Before a federate knows about an object instance, all of the instance attributes of that object instance are in the Attributes Out-of-Scope state with regard to the federate. (see Figure 14)

Rationale: The following two examples demonstrate the use of item 1:

Fed1 enables the Object Class Relevance Advisory switch or it is already enabled as a result of settings in the FDD.
Fed2 subscribes to object class A.B, attribute Y, denoted A(Y).
Fed1 publishes A.B(X). According to item 1 above, before federate 1 began publishing A.B(X), it was in a state such that registration of new object instances of object class A.B was not advised. Upon publishing A.B(X), it remains in a state such that registration of new object instances of object class A.B is not advised. Therefore, fed1 shall not receive a Stop Registration For Object Class advisory because the conditions covering that advisory have not changed at federate 1.

Alternatively,
Fed1 enables the Object Class Relevance Advisory switch or it is already enabled as a result of settings in the FDD.
Fed2 subscribes to A(X).
Fed1 publishes A(X). According to item 1 above, before federate 1 began publishing A(X), it was in a state such that registration of new object instances of object class A was not advised. Then, upon publishing A(X), it becomes in a state such that

registration of new object instances of object class A is advised. Because the conditions covering the advisory have changed, fed1 receives a Start Registration For Object Class † service invocation for object class A.

The following two examples demonstrate the use of item 2:
Fed1 enables the Interaction Relevance Advisory switch or it is already enabled as a result of settings in the FDD.
Fed2 subscribes to interaction class A.B.
Fed1 publishes interaction class A.C. According to item 2 above, before federate 1 began publishing interaction class A.C, it was in a state such that interactions of class A.C were not relevant to any other federate in the federation execution. Then, upon publishing interaction class A.C, it remained in a state such that interactions of class A.C are not relevant to other federates in the federation execution. Therefore, because the conditions covering that advisory have not changed at fed1, fed1 shall not receive a Turn Interactions Off † service invocation for interaction class A.C.

Alternatively,
Fed1 enables the Interaction Relevance Advisory switch or it is already enabled as a result of settings in the FDD.
Fed2 subscribes to interaction class A.
Fed1 publishes interaction class A. According to item 2 above, before federate 1 began publishing interaction class A, it was in a state such that interactions of class A were not relevant to other federates in the federation execution. Then, upon publishing interaction class A, it becomes in a state such that interactions of class A are relevant to other federates in the federation execution. Because the conditions covering the advisory have changed, fed1 receives a Turn Interactions On † service invocation for interaction class A.

The following is an example of the use of item 3 (ignoring scope for now):
Suppose fed1 and fed2 are both publishing and subscribing to A(x), but not A(y). The Attribute Relevance advisory switch is enabled at both federates.
Fed1 registers an instance of class A, A1, which fed2 is expected to discover.
As a result of registering A1, fed1 owns instance attribute x of A1. (call it A1(x)), but A1(y) is unowned.
According to item 3 above, before fed1 became the owner of A1(x) fed1 was in the Updates Turned Off state with regard to both A1(x) and A1(y). Upon becoming the owner of A1(x), fed1 enters the Updates Turned On state with regard to A1(x), and remains in the Updates Turned Off state with regard to A1(y). Because the conditions covering the advisory have changed for A1(x) but not A1(y), fed1 should get a Turn Updates On for A1(x), but it should not get any advisories for A1(y).

The following is also an implication of item 3:
Continuing with the previous example, fed1 transfers ownership of A1(x) to fed2.
According to item 3 above, before fed2 became the owner of A1(x), it was in the Updates Turned Off state with regard to A1(x). Upon becoming the owner of A1(x), fed2 enters the Updates Turned On state with regard to A1(x) and so receives a Turn Updates On † service invocation for A1(x).
Fed2 unsubscribes to A(x).
Fed2 transfers ownership of A1(x) back to fed1.
According to item 3 above, before fed1 became the owner of A1(x) it was in the Turn

Updates Off state with respect to A1(x). Upon becoming the owner of A1(x), fed1 is still in the Updates Turned Off state with respect to A1(x) because no other federate is subscribed to A(x). So, even though the last advisory that fed1 had received with respect to A1(x) was Turn Updates On, and now the conditions covering that advisory have changed such that fed1 is now in the Updates Turned Off state with respect to A1(x), according to item 3 above, fed1 will not receive a Turn Updates Off † callback for A1(x).

The following is an example of the use of item 4 in which the federate begins to know an object instance:
Suppose fed1 and fed2 are both publishing and subscribing to A(x), but not A(y). The Attribute Scope Advisory switch is enabled at both federates.
Fed1 registers an instance of this class, A1, which fed2 is expected to discover. According to item 4 above, before fed2 knew about A1, all of the instance attributes of A1 were in the Attributes Out-of-Scope state with regard to fed2; upon A1 becoming known by fed2, instance attribute A1(x) moves into the Attribute-In-Scope state at fed2 and instance attribute A1(y) remains in the Attribute-Out-of-Scope state at fed2, because A1(x) is owned by fed1 but A1(y) is not owned by any federate. Therefore, according to item 4 above, fed2 shall get an Attributes In Scope † callback only for A1(x).

1.4.8.2 Service 10.9: Get Parameter Name

Interpretation 1

Exception (c) should have a “not” in it. It should read “The parameter is **not** an available parameter of the interaction class.”

1.4.8.3 Service 10.30: Get Dimension Handle Set

Interpretation 1

There shall be an additional pre-condition to this service that says, “The region was either created by the invoking joined federate using the Create Region Service or it was conveyed to the invoking joined federate in a Set of Sent Region Designators argument.” If the region designator specified is not the designator of a region that was either created by the invoking federate or conveyed to it in a Set of Sent Region Designators argument, the “Invalid region” exception shall be generated.

Rationale: A federate shall not be able to invoke the Get Dimension Handle Set service on a region designator that it received by any means other than as a result of creating the region itself, or having had the region realization (copy) designator passed to it in a conveyed Set of Sent Region Designators argument of either a reflect or a received interaction. It is undefined as to what behavior would be expected if a federate were to receive a region realization designator by any other means and invoke the Get Dimension Handle Set service on it.

1.4.8.4 Service 10.31: Get Range Bounds

Interpretation 1

There shall be a precondition of this service that says, “All supplied region designators are designators of either region specifications or of region realization copies.” If a region designator specified is not a designator of a region specification or of a region realization copy, the “Invalid region” exception shall be generated.

Rationale: If the designator of a region template, instead of a designator of either a region specification or a region realization copy, is used as an argument to this service, then this service would not be able to return the range bounds for those dimensions of the region template that have not yet been set.

Interpretation 2

There shall be an additional pre-condition to this service that says, “The region was either created by the invoking joined federate using the Create Region Service or it was conveyed to the invoking joined federate in a Set of Sent Region Designators argument. If the region designator specified is not the designator of a region that was either created by the invoking federate or conveyed to it in a Set of Sent Region Designators argument, the “Invalid region” exception shall be generated.”

Rationale: A federate shall not be able to invoke the Get Range Bounds service on a region specification designator that it received by any means other than as a result of creating the region itself, or having had the region specification (copy) designator passed to it in a conveyed Set of Sent Region Designators argument of either a reflect or a received interaction. It is undefined as to what behavior would be expected if a federate were to receive a region specification designator by any other means and invoke the Get Range Bounds service on it.

1.4.9 Management Object Model Interpretations

1.4.9.1 11.1: MOM Overview

Interpretation 1

Some attribute definitions in Table 16 indicate that under certain circumstances the value of a MOM attribute shall be null. For example, according to Table 16, if no saves have occurred, the value of the HLAlastSaveName attribute shall be null. By “null” the expectation is that the attribute/value pair set will be present in the reflect, but the value will be an empty (zero-length) array.

1.4.9.2 11.4.1: Normal RTI MOM administration: item (g)

Interpretation 1

Item (g) shall refer to Table 6 instead of Table 4.

Rationale: Table 6, not Table 4, is the MOM attribute table.

Interpretation 2

Clause 11.4.1 (d) states that, "When sending an interactions of one of the leaf classes in Table 5, the RTI shall always supply all parameters listed in Table 7 for that interaction class, and no more." However, there exist the following exceptional cases in which the RTI shall not supply all the parameters:

- The HLAfederate parameter of the HLAmanger.HLAfederate.HLAreport.HLAreportSynchronizationPoints interaction shall not be supplied.
- The HLAfederate parameter of the HLAmanger.HLAfederate.HLAreport.HLAreportSynchronizationPointStatus interaction shall not be supplied.
- The HLAknownClass parameter of the HLAmanger.HLAfederate.HLAreport.HLAreportObjectInstanceInformation interaction shall not be supplied if the HLAfederate parameter of this interaction specifies a joined federate that does not know the object instance specified by the HLAobjectInstance parameter of this interaction.

Rationale: The value of the HLAfederate parameter is not relevant to the other information that is provided in the HLAmanager.HLAfederate.HLAreport.HLAreportSynchronizationPoints or the HLAmanager.HLAfederate.HLAreport.HLAreportSynchronizationPointStatus interactions. These interactions are more like HLAmanger.HLAfederation than HLAmanager.HLAfederate interactions. There is no reason that the RTI should supply the HLAfederate parameter.

If a federate does not know an object instance, then that object instance has no known class at that federate. Therefore, there is no valid value of the HLAknownClass parameter of an HLAmanger.HLAfederate.HLAreport.HLAreportObjectInstanceInformation interaction that can be sent for this federate and object instance.

Interpretation 3

Some parameter definitions in Table 17 indicate that under certain circumstances the value of a MOM interaction parameter shall be null. For example, according to Table 17, if the specified service does not normally return a value, then the HLAreturnedArgument parameter of the HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation interaction shall be null. Also, if the value of the HLAsuccessIndicator parameter of the HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation interaction is HLAtrue, then the value of the HLAexception parameter shall be null. By "null" the expectation is that the parameter/value pair set will be present in the interaction, but the value will be an empty (zero-length) array.

Interpretation 4

Unless specifically noted otherwise in Table 17, when a federate sends an interaction, it shall always supply all pre-defined parameters that are available at that interaction class and no more, with the following exceptions:

- The HLAfederate parameter of the HLAmanger.HLAfederate.HLArequest.HLArequestSynchronizationPoints interaction is not required.
- The HLAfederate parameter of the HLAmanger.HLAfederate.HLArequest.HLArequestSynchronizationPointStatus interaction is not required.
- A federate shall not be required to supply parameters of any HLAmanager.HLAfederate.HLAservice interaction that correspond to optional arguments of the HLA service that the HLAmanager.HLAfederate.HLAservice interaction is intended to cause to be invoked on behalf of another federate. (For example, HLAattributeList is not a required parameter of the HLAmanager.HLAfederate.HLAservice.HLAunpublishObjectClassAttributes interaction because the set of attribute designators argument of the Unpublish Object Class Attributes service is an optional argument to that service).
- A federate shall be required to supply either the HLAautoProvide parameter or the HLAconveyRegionDesignatorSets parameter (or both) of the HLAmanager.HLAfederation.HLAadjust.HLAsetSwitches interaction.

Rationale: While section 11.4.1 specifies what parameters shall be supplied when the RTI sends interactions, it does not discuss what parameters shall be supplied when a federate sends MOM interactions. In order for the RTI to be able to send an HLAmanager.HLAfederate.HLAreport.HLAreportMOMexception interaction when a MOM interaction is sent without all the necessary parameters, it must be well-defined as to what the necessary parameters for each interaction are.

The rationale for why the HLAfederate parameter of the HLAmanager.HLAfederate.HLArequest.HLArequestSynchronizationPoints and the HLAmanager.HLAfederate.HLArequest.HLArequestSynchronizationPointStatus interactions is not required is that these interactions request federation-wide information rather than federate-specific information, so the value of the HLAfederate parameter is not relevant to these interactions.

1.4.9.3 Table 6: MOM attribute table: HLAfederateState

Interpretation 1

The HLAfederateState attribute of the HLAmanger.HLAfederate object class is of update type *Conditional* and its update condition is *Service Invocation*. This means that if a service invocation occurs that causes the HLAfederateState of an HLAmanger.HLAfederate object instance to change value, then the corresponding instance attribute will be updated. An exception to this occurs when a federate's state changes from that of *Active Federate* to that of *Federate Restore In Progress*. In this case, no updates are expected. In fact, the MOM is not expected to update any instance

attribute values after the first Federation Restore Begun † callback is invoked at any federate in the federation execution and before the last Federation Restored † callback is invoked at some federate for a given federation restoration.

Rationale: When a federate is in the Federate Restore In Progress state, the identity of that federate, being in a possible state of flux, is undefined. When at least one federate in the federation execution is in the Federate Restore In Progress state, it does not make sense for the RTI to maintain the values of instance attributes of HLAmanager.HLAfederate object instances, because it is not clear to which federate any given HLAmanager.HLAfederate object instance corresponds, nor is it clear at which federate an update to such an instance attribute should be reflected. The state of each HLAmanger.HLAfederate object instance is in flux during a restore. Only after all federates have restored successfully and moved back into the Active Federate state should the RTI resume maintaining the values of instance attributes of HLAmanager.HLAfederate object instances.

1.4.9.4 Table 15: MOM interaction class definitions table: HLArequestSubscriptions

Interpretation 1

The definition of the HLAmanager.HLAfederate.HLArequest.HLArequestSuscriptions interaction says that this interaction shall result in one interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportInteractionSubscription and one interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportObjectClassSubscription for each object class published. Instead, it should say that it shall result in one interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportInteractionSubscription and one interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportObjectClassSubscription for each **different combination of (object class, passive subscription indicator) values that are subscribed**.

In other words, if a federate is subscribed to a given object class and class attribute with the same passive/active subscription indicator value either with multiple DDM subscriptions or with one or more DDM subscriptions and a DM subscription, that (object class, attribute, active/passive indicator) triple should only appear in one HLAmanager.HLAfederate.HLAreport.HLAreportObjectClass Subscription interaction that is sent. However, if a federate is subscribed to a given object class and class attribute with different passive/active subscription indicators (at least once actively and at least once passively), either with multiple DDM subscriptions or with one or more DDM subscriptions and a DM subscription, that (object class, attribute, active/passive indicator) triple should appear in two separate HLAmanager.HLAfederate.HLAreport.HLAreportObjectClass Subscription interactions that are sent, one of which has an HLAactive parameter value of HLAtrue, and one of which has an HLAactive parameter value of HLAfalse.

In addition, the HLAnumberOfClasses parameter shall represent the count of the number of different (object class, active/passive subscription indicator) values being reported. This number shall not exceed twice the number of different object classes that are subscribed.

Similarly, if a federate is subscribed to a given interaction class with the same active/passive subscription indicator value with both a DDM subscription and a DM subscription, that (interaction class, active/passive indicator) pair should appear only once in the HLAmanager.HLAfederate.HLAreport.HLAreportInteractionSubscription interaction that is sent. However, if a federate is subscribed to a given interaction class with different active/passive subscription indicators (once actively and once passively), once with a DDM subscription and once with a DM subscription, that (interaction class, active/passive indicator) pair should appear twice in the HLAmanager.HLAfederate.HLAreport.HLAreportInteractionSubscription interaction that is sent, once with an HLAactive parameter value of HLAtrue, and once with an HLAactive parameter value of HLAfalse.

Rationale: The change of the word "published" to "subscribed is a correction of a typographical error.

The change that the HLArequestSubscriptions interaction shall result in one interaction of class HLAreportObjectClassSubscriptionfor each different combination of (object class, passive subscription indicator) values that are subscribed by the federate is required in order to enable the information in the HLAreportObjectClass subscription interaction to, as specified in the Table 17 parameter definitions for the HLAnumberofClasses and HLAinteractionClassList parameters, "reflect related DDM usage."

Each HLAreportObjectClassSubscription interaction must, according to Table 17, contain four parameters: HLAnumberOfClasses, HLAobjectClass, HLAactive, and HLAattributeList. When a federate subscribes to object class attributes using only DM subscriptions, all attributes that are subscribed at a given class must necessarily be all subscribed with the same passive subscription indicator value (either passive or active). However, when a federate subscribes to object class attributes using DDM subscriptions, it is possible for the federate to be subscribed to the same attribute at a given object class both passively and actively (as long as they are subscribed with different regions). The parameter information present in the HLAreportObjectClassSubscription interaction is not flexible enough to both meet the constraint that at most one interaction of the class HLAmanager.HLAfederate.HLAreport.HLAreportObjectClassSubscription shall be sent for each object class subscribed, and to accurately convey whether these subscriptions are either active, passive, or both active and passive.

The HLAnumberOfClasses parameter shall represent the count of the number of different (object class, active/passive subscription indicator) values being reported because this parameter is used to indicate to the federate how many HLAreportObjectClassSubscription interactions to expect from the RTI.

1.4.9.5 Table 15: MOM interaction class definitions table: HLAreportObjectInstancesUpdated:

Interpretation 1

This interaction shall report the number of object instances (by registered class of the object instances) for which the joined federate has **successfully** invoked the *Update Attribute Values* service.

1.4.9.6 Table 15: MOM interaction class definitions table:HLAreportSynchronizationPointStatus

Interpretation 1

One interaction of class HLAmanger.HLAfederate.HLAreport.HLAreportSynchronizationPointStatus shall be sent by the RTI for each active synchronization point in the federation execution. If there are no active synchronization points in the federation execution, no HLAmanger.HLAfederate.HLAreport.HLAreportSynchronizationPointStatus interaction shall be sent.

Rationale: Each HLAmanger.HLAfederate.HLAreport.HLAreportSynchronizationPointStatus interaction reports on the status of only one synchronization point. Because the HLAmanger.HLAfederate.HLArequest.HLArequestSynchronizationPointStatus interaction does not include a HLAsyncPointName parameter that could be used to specify which synchronization point for which a report is requested, one HLAmanger.HLAfederate.HLAreport.HLAreportSynchronizationPointStatus interaction shall be sent for each active synchronization point in the federation execution. A federate is able to use the HLAmanger.HLAfederate.HLArequest.HLArequestSynchronizationPoints interaction to receive a report of all active synchronization points in the federation execution, so if a federate invokes the HLAmanager.HLAfederate.HLArequest.HLArequestSynchronizationPointStatus interaction when there are no active synchronization points, it is allowable for that federate to fail to receive an HLAmanger.HLAfederate.HLAreport.HLAreportSynchronizationPointStatus interaction in response.

1.4.9.7 Table 16: MOM attribute definitions table: HLAFDDID

Interpretation 1

The HLAFDDID attribute of the MOM HLAmanager.HLAfederation object class is defined as the “identifier associated with the FDD used in the relevant Create Federation Execution service invocation.” In particular, this identifier shall be the same as the FOM document designator argument that was supplied in the Create Federation Execution service when the federation execution was created. However, all of the path-

specific information shall have been removed from the designator and, if this designator took the form of a URL, all of the URL-specific information shall also have been removed.

There is also an HLAFDDID attribute of the MOM HLAmanager.HLAfederate object class that is defined as the “identifier associated with the FDD used in the joined federate.” This identifier shall be the same as the FOM document designator argument that was supplied in the Create Federation Execution service when the federation execution was created.

1.4.9.8 Table 16: MOM attribute definitions table: HLAreflectionsReceived

Interpretation 1

The HLAreflectionsReceived attribute shall have as value the total number of times the Reflect Attribute Values † service has been invoked at the joined federate (as opposed to the number of instance attribute value reflections have been received at the joined federate).

1.4.9.9 Table 16: MOM attribute definitions table: HLAupdatesSent

Interpretation 1

The HLAupdatesSent attribute shall have as value the total number of times the Update Attribute Values † service has successfully been invoked by the joined federate (as opposed to the number of instance attribute values that have been updated by the joined federate).

1.4.9.10 Table 16: MOM attribute definitions table: HLAlastSaveTime

Interpretation 1

According to the attribute definitions inTable 16, the HLAlastSaveTime value shall not be defined if no timed saves have occurred. By “not defined” the expectation is that the attribute/value pair set will be present in the reflect, but the value will be an empty (zero-length) HLAlogicalTime array.

Interpretation 2

The HLAlastSaveTime attribute shall have the value of the time of the last save, not of the last timed save. If the last save was not a timed save, then the HLAlastSaveTime attribute value shall be an empty HLAlogicalTime array to indicate that the value of the HLAlastSaveTime attribute is undefined.

Rationale: The value of the HLAlastSaveName attribute should correspond to the value of the HLAlastSaveTime attribute. The way the definition of HLAlastSaveTime is worded in the specification, if a timed save occurs, followed by an untimed save, then the value of HLAlastSaveName would not correspond with the value of

HLAlastSaveTime, which could prove astonishing to a user. Therefore, in order to ensure that these two values always correspond to the same save, if the last save is an untimed save, then the value of the HLAlastSaveTime attribute will not be defined.

1.4.9.11 Table 16: MOM attribute definitions table: HLAnextSaveTime

Interpretation 1

According to the attribute definitions in Table 16, the HLAnextSaveTime value shall not be defined if no timed saves are scheduled. By “not defined” the expectation is that the attribute/value pair set will be present in the reflect, but the value will be an empty (zero-length) HLAlogicalTime array.

1.4.9.12 Table 16: MOM attribute definitions table: HLAobjectInstancesUpdated

Interpretation 1

The HLAobjectInstancesUpdated attribute shall be defined as the total number of object instances for which the joined federate has **successfully** invoked the *Update Attribute Values* service.

1.4.9.13 Table 16: MOM attribute definitions table: HLAobjectInstancesDeleted

Interpretation 1

The HLAobjectInstancesDeleted attribute shall be defined as the total number of times that the *Delete Object Instance* service was **successfully** invoked by the joined federate since the federate joined the federation.

1.4.9.14 Table 16: MOM attribute definitions table: HLAobjectInstancesRegistered

Interpretation 1

The HLAobjectInstancesRegistered attribute shall be defined as the total number of times that the *Register Object Instance* service **and** *the Register Object Instance With Re*gion service **were successfully** invoked by the joined federate since the federate joined the federation.

1.4.9.15 Table 16: MOM attribute definitions table: HLAobjectInstancesDiscovered

Interpretation 1

The value of the HLAobjectInstancesDiscovered attribute shall include multiple invocations of the *Discover Object Instance* † service for a given object instance that may occur as a result of invocation of the Local Delete Object Instance service at a federate.

1.4.9.16 Table 16: MOM attribute definitions table: HLAtimeGrantedTime and HLAtimeAdvancingTime

Interpretation 1

These attributes are defined as the wall-clock time duration that the federate has spent in a given state since the last time the attributes were updated. It does not specify what the value of these attributes should be if they have not yet been updated. The first time that the HLAtimeGrantedTime and the HLAtimeAdvancingTime attributes are updated, their values shall be zero to indicate that they have never before been updated.

1.4.9.17 Table 17: MOM parameter definitions table: HLAreportPeriod

Interpretation 1

If no interaction of class HLAmanager.HLAfederate.HLAadjust.HLAsetTiming has been sent, then no periodic updates of MOM attribute values shall be generated.

Rationale: If no interaction of class HLAmanager.HLAfederate.HLAadjust.HLAsetTiming has been sent, then the value of the HLAreportPeriod is not defined. It makes sense to interpret this value to be zero (which means that periodic updates will not occur) unless and until this HLAreportPeriod value is explicitly set by the invocation of an HLAmanager.HLAfederate.HLAadjust.HLAsetTiming interaction.

1.4.9.18 Table 17: MOM interaction subclass HLAmanager.HLAfederate.HLAreport.HLAreportUpdatesSent

Interpretation 1

According to the definition of the HLAupdateCounts parameter in the HLAReportUpdatesSent interaction subclass given in Table 17, this parameter consists of a list of update counts, each of which consists of an object class handle and “the number of updates sent of that class.” The question of what makes an update be of one class as opposed to another is answered in Table 15 (HLAreportUpdatesSent Interaction class definition): the class of an update is the registered class of the object instance of the update. However, it is not clear whether the number of updates is defined as the number of times the *Update Attribute Values* service was invoked by the

federate for all object instances of a given object class, or the number of instance attribute updates that were accomplished by the federate for all object instances of a given object class.

The expectation is that the number of updates is defined as the number of instance attribute updates. That is, if a federate has invoked the *Update Attribute Values* service only once, and in this service invocation were arguments for an object instance of class A and *n* instance attributes of type reliable and *m* instance attributes of type best-effort, then in response to an interaction of class Manager.Federate.Request.RequestUpdatesSent, two Manager.Federate.Report.ReportUpdatesSent interactions should be sent: one for transportation type reliable with an update count of *n* and one for transportation type best-effort with an update count of *m*. If that federate then invokes the *Update Attribute Values* service for an object instance of class A and one of the same instance attributes that was updated in the previous update of type reliable, then in response to an interaction of class Manager.Federate.Request.RequestUpdatesSent, two Manager.Federate.Report.ReportUpdatesSent interactions should be sent: one for transportation type reliable with an update count of *n+1* and one for transportation type best-effort with an update count of *m*.

Rationale: The update service is a service that acts on instance attributes, not on object instances. Similarly, transportation type is a property of instance attributes rather than object instances. The fact that updates to several different instance attributes of an object instance can be bundled together in a single Update Attribute Values service invocation is provided as a convenience to the programmer. The value of an update count should not depend on whether or not a federate chooses to combine certain instance attribute value updates together in a single call or perform these updates as separate Update Attribute Values service invocations.

Interpretation 2

If no updates of instance attributes of any object instances of any class for a given transportation type have been sent, then the RTI shall send an HLAmanager.HLAfederate.HLAreport.HLAreportUpdatesSent interaction for that transportation type. However, no HLAobjectClassBasedCount elements at all should appear in the HLAobjectClassBasedCount array for that interaction of that transportation type. In other words, the HLAreportUpdatesSent interaction that is sent for that transportation type will have an empty HLAobjectClassBasedCount array. (This is illustrated by interaction 2 in the example below.)

If no updates of instance attributes of any object instances of a given class for a given transportation type have been sent, then no HLAobjectClassBasedCount element for that object class should be in the HLAobjectClassBasedCount array of the HLAmanager.HLAfederate.HLAreport.HLAreportUpdatesSent interaction for that transportation type. (This is illustrated by interaction 1 below.) Consider the following example:

Suppose there are 3 classes defined in the FDD, A, A.B, and A.C.

Suppose there are 2 transportation types available for use.

Suppose that only the following 2 updates were sent:

Update of an object instance of class A, reliable attribute x

Update of an object instance of class A.B, reliable attribute x and reliable attribute z.

Then, 2 HLAreportUpdatesSent interactions would be sent in response to an HLARequestUpdatesSent interaction, and those interactions would be as follows:

1. An interaction with 2 parameters: transportation type Reliable, and an HLAobjectClassBasedCount array with 2 HLAobjectClassBasedCount elements in it; one HLAobjectClassBasedCount element would be (class A, 1) the other element would be (class A.B, 2). (There would be no HLAobjectClassBasedCount element for class A.C in this array.)
2. An interaction with 2 parameters: transportation type Best effort, and an HLAobjectClassBasedCount array with no elements in it.

1.4.9.19 Table 17: MOM interaction subclass HLAmanager.HLAfederate.HLAreport.HLAreportReflectionsReceived

Interpretation 1

According to the definition of the HLAreflectCounts parameter in the HLAReportReflectionsReceived interaction subclass, this parameter consists of a list of reflection counts, each of which consists of an object class handle and “the number of reflections received of that class.” This wording is ambiguous regarding the question of whether the number of reflections is defined as the number of times the *Reflect Attribute Values* † service was invoked at the federate for all object instances of a given object class, or the number of instance attribute value reflections by the federate for all object instances of a given object class.

The expectation is that the number of reflections is defined as the number of instance attribute value reflections. That is, if a federate has received the *Reflect Attribute Values* † service invocation twice, and in one of these service invocations were arguments for an object instance of class A and *n* instance attributes of type reliable, and in another of these service invocations were arguments for an object instance of class A and *m* instance attributes of type best-effort, then in response to an interaction of class Manager.Federate.Request.RequestReflectionsReceived, two Manager.Federate.Report.ReportReflectionsReceived interactions should be sent: one for transportation type reliable with a reflect count of *n* for object class A, and one for transportation type best-effort with a reflection count of *m* for object class A. Furthermore, if that federate receives an additional *Reflect Attribute Values* † service invocation for an object instance of class A that contains a single attribute/value pair as argument, and the attribute is a reliable attribute that had also had a value reflected previously, then in response to an interaction of class Manager.Federate.Request.RequestReflectionsReceived, two Manager.Federate.Report.ReportReflectionsReceived interactions should be sent: one for transportation type reliable with a reflect count of *n + 1* for object class A, and one for transportation type best-effort with a reflection count of *m* for object class A.

Rationale: The rationale for this interpretation is analogous to the rationale for the MOM Table 17: interaction subclass HLAmanager.HLAfederate.HLAreport.HLAreportUpdatesSent interpretation. As with the Update Attribute Values service, the Reflect Attribute Values † service is a service that acts on instance attributes, not on object instances. Similarly, transportation type is a property of instance attributes rather than of object instances.

Interpretation 2

If no reflects of instance attributes of any object instances of any class for a given transportation type have been received, then the RTI shall send an HLAmanager.HLAfederate.HLAreport.HLAreportReflectionsReceived interaction for that transportation type. However, no HLAobjectClassBasedCount elements at all shall appear in the HLAobjectClassBasedCount array for that interaction of that transportation type. In other words, the HLAreportReflectionsReceived interaction that is sent for that transportation type shall have an empty HLAobjectClassBasedCount array. (This is illustrated by interaction 2 in the example below.)

If no reflects of instance attributes of any object instances of a given class for a given transportation type have been received, then no HLAobjectClassBasedCount element for that object class shall be in the HLAobjectClassBasedCount array of the HLAmanager.HLAfederate.HLAreport.HLAreportReflectionsReceived interaction for that transportation type. (This is illustrated by interaction 1 below.)

Consider the following example:

Suppose there are 3 classes defined in the FDD, A, A.B, and A.C.

Suppose there are 2 transportation types available for use.

Suppose that only the following 2 reflects were received:

Reflect of an object instance of class A, reliable attribute x

Reflect of an object instance of class A.B, reliable attribute x and reliable attribute z.

Then, 2 HLAreportReflectionsReceived interactions would be sent in response to a HLARequestReflectionsReceived interaction, and those interactions would be as follows:

1. An interaction with 2 parameters: transportation type Reliable, and an HLAobjectClassBasedCount array with 2 HLAobjectClassBasedCount elements in it; one HLAobjectClassBasedCount element would be (class A, 1) the other element would be (class A.B, 2). (There would be no HLAobjectClassBasedCount element for class A.C in this array.)

2. An interaction with 2 parameters: transportation type Best effort, and an HLAobjectClassBasedCount array with no elements in it.

1.4.9.20 Table 17: MOM use of HLAobjectClassBasedCounts array datatype in zero-value HLAobjectClassBasedCount cases

Interpretation 1

This interpretation is a generalization of each of the Interpretation 2 stated above for the HLAmanager.HLAfederate.HLAreport.HLAreportUpdatesSent and the HLAmanager.HLAfederate.HLAreport.HLAreportReflectionsReceived interactions. In all MOM interactions that have a parameter of type HLAobjectClassBasedCounts, if an HLAobjectClassBasedCount element of the HLAobjectClassBasedCounts array would have a value (object class, 0), the HLAobjectClassBasedCount element shall not be present in the HLAobjectClassBasedCounts array. In other words, only HLAobjectClassBasedCount elements that have positive counts shall be present in an HLAobjectClassBasedCounts array. From this, it follows that if all object class counts have a zero value, then the HLAobjectClassBasedCounts array shall not have any elements in it; it shall be an empty HLAobjectClassBasedCounts array. This interpretation affects the following MOM interactions:

- HLAmanager.HLAfederate.HLAreport.HLAreportObjectInstancesThatCanBeDeleted
- HLAmanager.HLAfederate.HLAreport.HLAreportObjectInstancesUpdated
- HLAmanager.HLAfederate.HLAreport.HLAreportObjectInstancesReflected
- HLAmanager.HLAfederate.HLAreport.HLAreportUpdatesSent (see its Interpretation 2)
- HLAmanager.HLAfederate.HLAreport.HLAreportReflectionsReceived (see its Interpretation 2)

1.4.9.21 Table 17: MOM use of HLAinteractionCounts array datatype in zero-value HLAinteractionCount cases

Interpretation 1

In all MOM interactions that have a parameter of type HLAinteractionCounts, if an HLAinteractionCount element of the HLAinteractionCounts array would have a value (interaction class, 0), the HLAinteractionCount element shall not be present in the HLAinteractionCounts array. In other words, only HLAinteractionCount elements that have positive counts shall be present in an HLAinteractionCounts array. From this, it follows that if all interaction class counts have a zero value, then the HLAinteractionCounts array shall not have any elements in it; it shall be an empty HLAinteractionCounts array. This interpretation affects the following MOM interactions:

- HLAmanager.HLAfederate.HLAreport.HLAreportInteractionsSent
- HLAmanager.HLAfederate.HLAreport.HLAreportInteractionsReceived

1.4.9.22 Table 17: MOM HLAreportServiceInvocation: HLAreturnedArguments parameter

Interpretation 1

In table 17, the HLAreturnedArguments parameter is erroneously listed as singular ("HLAreturnedArgument") instead of plural. In order to be consistent with Table 7, the MOM parameter table, this parameter should be named "HLAreturnedArguments."

1.4.9.23 Table 17: MOM HLAreportMOMexception: HLAservice parameter

Interpretation 1

The definition of the HLAservice parameter says "Name of the service interaction that had a problem or raised an exception." In the case in which the HLAreportMOMexception interaction is sent by the RTI because a service interaction (an interaction that imitates a federate's invocation of an HLA service) was sent and not all of the service's pre-conditions are met, the value of this parameter shall be the name of the HLAinteractionRoot.HLA.Manager.HLAfederate.HLAservice interaction that was sent. In the case in which the HLAreportMOMexception interaction is sent by the RTI because a MOM interaction without all of the necessary parameters was sent, the value of this parameter shall be the name of the class of the interaction that was sent.

Rationale: In the second case, the case in which the HLAreportMOMexception interaction is sent by the RTI because a MOM interaction without all of the necessary parameters was sent, there is no HLA service interaction involved. Providing the name of the class of interaction that was sent that caused the HLAreportMOMexception invocation at least provides information to the sending federate as to what the offending class of the sent interaction was.

Interpretation 2

The name of the interaction class provided shall always be fully qualified, as defined in the OMT Specification, so as to avoid potential ambiguities.

OMG IDL A

A.1 Complete IDL Definitions

The following is the complete IDL for the specification. A compilable form of this IDL is available as mfg/2001-10-02.

Note – The numbers included as comments with each method on the ambassador interfaces are the corresponding service numbers in the Interface Specification. See Section 1.2, “Specifications Incorporated by Reference,” on page 1-3 for additional information.

```
//File: DistributedSimulation.idl
//this file is available as OMG document dtc/2002-05-03

#ifndef __DISTRIBUTED_SIMULATION_DEFINED
#define __DISTRIBUTED_SIMULATION_DEFINED

#pragma prefix "omg.org"

module DistributedSimulation
{

 #define RTI_EXCEPT(A) \
 exception A { \
  string reason; \
 };

 RTI_EXCEPT(AsynchronousDeliveryAlreadyDisabled)
 RTI_EXCEPT(AsynchronousDeliveryAlreadyEnabled)
 RTI_EXCEPT(AttributeAcquisitionWasNotCanceled)
 RTI_EXCEPT(AttributeAcquisitionWasNotRequested)
 RTI_EXCEPT(AttributeAlreadyBeingAcquired)
```

RTI_EXCEPT(AttributeAlreadyBeingDivested)
RTI_EXCEPT(AttributeAlreadyOwned)
RTI_EXCEPT(AttributeDivestitureWasNotRequested)
RTI_EXCEPT(AttributeNotDefined)
RTI_EXCEPT(AttributeNotOwned)
RTI_EXCEPT(AttributeNotPublished)
RTI_EXCEPT(AttributeNotRecognized)
RTI_EXCEPT(AttributeNotSubscribed)
RTI_EXCEPT(AttributeRelevanceAdvisorySwitchIsOff)
RTI_EXCEPT(AttributeRelevanceAdvisorySwitchIsOn)
RTI_EXCEPT(AttributeScopeAdvisorySwitchIsOff)
RTI_EXCEPT(AttributeScopeAdvisorySwitchIsOn)
RTI_EXCEPT(AttributeSetRegionSetPairListFactory)
RTI_EXCEPT(BadInitializationParameter)
RTI_EXCEPT(CouldNotDecode)
RTI_EXCEPT(CouldNotDiscover)
RTI_EXCEPT(CouldNotInitiateRestore)
RTI_EXCEPT(CouldNotOpenFDD)
RTI_EXCEPT(DeletePrivilegeNotHeld)
RTI_EXCEPT(ErrorReadingFDD)
RTI_EXCEPT(FederateAlreadyExecutionMember)
RTI_EXCEPT(FederateHasNotBegunSave)
RTI_EXCEPT(FederateInternalError)
RTI_EXCEPT(FederateNotExecutionMember)
RTI_EXCEPT(FederateOwnsAttributes)
RTI_EXCEPT(FederatesCurrentlyJoined)
RTI_EXCEPT(FederateServiceInvocationsAreBeingReportedViaMOM)
RTI_EXCEPT(FederateUnableToUseTime)
RTI_EXCEPT(FederationExecutionAlreadyExists)
RTI_EXCEPT(FederationExecutionDoesNotExist)
RTI_EXCEPT(IllegalName)
RTI_EXCEPT(IllegalTimeArithmetic)
RTI_EXCEPT(InitializeNeverInvoked)
RTI_EXCEPT(InitializePreviouslyInvoked)
RTI_EXCEPT(InteractionClassNotDefined)
RTI_EXCEPT(InteractionClassNotPublished)
RTI_EXCEPT(InteractionClassNotRecognized)
RTI_EXCEPT(InteractionClassNotSubscribed)
RTI_EXCEPT(InteractionParameterNotDefined)
RTI_EXCEPT(InteractionParameterNotRecognized)
RTI_EXCEPT(InteractionRelevanceAdvisorySwitchIsOff)
RTI_EXCEPT(InteractionRelevanceAdvisorySwitchIsOn)
RTI_EXCEPT(InTimeAdvancingState)
RTI_EXCEPT(InvalidAttributeHandle)
RTI_EXCEPT(InvalidDimensionHandle)
RTI_EXCEPT(InvalidFederateHandle)
RTI_EXCEPT(InvalidInteractionClassHandle)
RTI_EXCEPT(InvalidLogicalTime)
RTI_EXCEPT(InvalidLookahead)
RTI_EXCEPT(InvalidMessageRetractionHandle)
RTI_EXCEPT(InvalidObjectClassHandle)

```
RTI_EXCEPT(InvalidOrderName)
RTI_EXCEPT(InvalidOrderType)
RTI_EXCEPT(InvalidParameterHandle)
RTI_EXCEPT(InvalidRangeBound)
RTI_EXCEPT(InvalidRegion)
RTI_EXCEPT(InvalidRegionContext)
RTI_EXCEPT(InvalidResignAction)
RTI_EXCEPT(InvalidServiceGroup)
RTI_EXCEPT(InvalidTransportationName)
RTI_EXCEPT(InvalidTransportationType)
RTI_EXCEPT(JoinedFederateIsNotInTimeAdvancingState)
RTI_EXCEPT(LogicalTimeAlreadyPassed)
RTI_EXCEPT(MessageCanNoLongerBeRetracted)
RTI_EXCEPT(NameNotFound)
RTI_EXCEPT(NoAcquisitionPending)
RTI_EXCEPT(NoRequestToEnableTimeConstrainedWasPending)
RTI_EXCEPT(NoRequestToEnableTimeRegulationWasPending)
RTI_EXCEPT(ObjectClassNotDefined)
RTI_EXCEPT(ObjectClassNotPublished)
RTI_EXCEPT(ObjectClassNotRecognized)
RTI_EXCEPT(ObjectClassRelevanceAdvisorySwitchIsOff)
RTI_EXCEPT(ObjectClassRelevanceAdvisorySwitchIsOn)
RTI_EXCEPT(ObjectInstanceNameInUse)
RTI_EXCEPT(ObjectInstanceNameNotReserved)
RTI_EXCEPT(ObjectInstanceNotKnown)
RTI_EXCEPT(OwnershipAcquisitionPending)
RTI_EXCEPT(RegionDoesNotContainSpecifiedDimension)
RTI_EXCEPT(RegionInUseForUpdateOrSubscription)
RTI_EXCEPT(RegionNotCreatedByThisFederate)
RTI_EXCEPT(RequestForTimeConstrainedPending)
RTI_EXCEPT(RequestForTimeRegulationPending)
RTI_EXCEPT(RestoreInProgress)
RTI_EXCEPT(RestoreNotInProgress)
RTI_EXCEPT(RestoreNotRequested)
RTI_EXCEPT(RTIinternalError)
RTI_EXCEPT(SaveInProgress)
RTI_EXCEPT(SaveNotInitiated)
RTI_EXCEPT(SaveNotInProgress)
RTI_EXCEPT(SpecifiedSaveLabelDoesNotExist)
RTI_EXCEPT(SomeFederateJoinedToAnExecution)
RTI_EXCEPT(SynchronizationPointLabelNotAnnounced)
RTI_EXCEPT(TimeConstrainedAlreadyEnabled)
RTI_EXCEPT(TimeConstrainedIsNotEnabled)
RTI_EXCEPT(TimeRegulationAlreadyEnabled)
RTI_EXCEPT(TimeRegulationIsNotEnabled)
RTI_EXCEPT(UnableToPerformSave)
RTI_EXCEPT(UnknownName)

typedef sequence<octet> Encoding;
typedef sequence<octet> AttributeValue;
typedef sequence<octet> ParameterValue;
```

```
typedef sequence<octet> UserSuppliedTag;
typedef sequence<octet> PropertyKey;
typedef sequence<octet> PropertyValue;

//enums are prefixed because IDL doesn't scope the values
//to the enumeration definions

enum ResignAction {
  RA_UNCONDITIONALLY_DIVEST_ATTRIBUTES,
  RA_DELETE_OBJECTS,
  RA_CANCEL_PENDING_OWNERSHIP_ACQUISITIONS,
  RA_DELETE_OBJECTS_THEN_DIVEST,
  RA_CANCEL_THEN_DELETE_THEN_DIVEST,
  RA_NO_ACTION
};

enum SynchronizationPointFailureReason {
  SPFR_SYNCHRONIZATION_POINT_LABEL_NOT_UNIQUE,
  SPFR_SYNCHRONIZATION_SET_MEMBER_NOT_JOINED
};

enum SaveFailureReason {
  SFR_RTI_UNABLE_TO_SAVE,
  SFR_FEDERATE_REPORTED_FAILURE,
  SFR_FEDERATE_RESIGNED,
  SFR_RTI_DETECTED_FAILURE,
  SFR_SAVE_TIME_CANNOT_BE_HONORED
};

enum RestoreFailureReason {
  RFR_RTI_UNABLE_TO_RESTORE,
  RFR_FEDERATE_REPORTED_FAILURE,
  RFR_FEDERATE_RESIGNED,
  RFR_RTI_DETECTED_FAILURE
};

enum SaveStatus {
  SS_NO_SAVE_IN_PROGRESS,
  SS_FEDERATE_INSTRUCTED_TO_SAVE,
  SS_FEDERATE_SAVING,
  SS_FEDERATE_WAITING_FOR_FEDERATION_TO_SAVE
};

enum RestoreStatus {
  RS_NO_RESTORE_IN_PROGRESS,
  RS_FEDERATE_RESTORE_REQUEST_PENDING,
  RS_FEDERATE_WAITING_FOR_RESTORE_TO_BEGIN,
  RS_FEDERATE_PREPARED_TO_RESTORE,
  RS_FEDERATE_RESTORING,
  RS_FEDERATE_WAITING_FOR_FEDERATION_TO_RESTORE
};
```

```
  enum ServiceGroup {
    SG_FEDERATION_MANAGEMENT,
    SG_DECLARATION_MANAGEMENT,
    SG_OBJECT_MANAGEMENT,
    SG_OWNERSHIP_MANAGEMENT,
    SG_TIME_MANAGEMENT,
    SG_DATA_DISTRIBUTION_MANAGEMENT,
    SG_SUPPORT_SERVICES
  };

  //forward references
interface FederateAmbassador;

  //time representation
typedef double LogicalTime;
typedef double LogicalTimeInterval;

  //handles
  #define HANDLETYPE(A) \
interface A { \
    boolean equals(in A h); \
    long hash_code(); \
    string to_string(); \
    long encoded_length(); \
    Encoding encode(); \
  }; \
  interface A##Factory { \
    A decode(in Encoding anEncoding) \
      raises(CouldNotDecode, FederateNotExecutionMember); \
  };

  HANDLETYPE(FederateHandle)
  HANDLETYPE(ObjectClassHandle)
  HANDLETYPE(AttributeHandle)
  HANDLETYPE(InteractionClassHandle)
  HANDLETYPE(ParameterHandle)
  HANDLETYPE(ObjectInstanceHandle)
  HANDLETYPE(DimensionHandle)
  HANDLETYPE(RegionHandle)
  HANDLETYPE(MessageRetractionHandle)

  interface OrderType {
    boolean equals(in FederateHandle h);
    long hash_code();
    string to_string();
    long encoded_length();
    Encoding encode();
};

interface OrderTypeFactory {
```

```
    OrderType decode(in Encoding anEncoding);
  };

  interface TransportationType {
    boolean equals(in FederateHandle h);
    long hash_code();
    string to_string();
    long encoded_length();
    Encoding encode();
  };

  interface TransportationTypeFactory {
   TransportationType decode(in Encoding anEncoding);
  };

  //this module introduced to work around deficiency in Ada mapping
  module CompositeTypes {

    //composite types
    typedef sequence<FederateHandle>  FederateHandleSet;
    typedef sequence<AttributeHandle> AttributeHandleSet;
    typedef sequence<DimensionHandle> DimensionHandleSet;
    typedef sequence<RegionHandle>    RegionHandleSet;

    struct FederateHandleSaveStatusPair {
      FederateHandle handle;
      SaveStatus status;
    };
    struct FederateHandleRestoreStatusPair {
      FederateHandle handle;
      RestoreStatus status;
    };

    typedef sequence<FederateHandleSaveStatusPair>    SaveStatusSequence;
    typedef sequence<FederateHandleRestoreStatusPair> RestoreStatusSequence;

    struct AttributeHandleValuePair {
      AttributeHandle handle;
      AttributeValue value;
    };

    typedef sequence<AttributeHandleValuePair>
      AttributeHandleValuePairSequence;

    struct ParameterHandleValuePair {
      ParameterHandle handle;
      ParameterValue value;
    };

    typedef sequence<ParameterHandleValuePair>
      ParameterHandleValuePairSequence;
```

```
    struct AttributeSetRegionSetPair {
      AttributeHandleSet attributes;
      RegionHandleSet    regions;
    };

    typedef sequence<AttributeSetRegionSetPair>
      AttributeSetRegionSetPairSequence;

    struct RangeBounds {
      long long lower;
      long long upper;
    };

    struct Property {
            wstring key;
      wstring value;
    };

    typedef sequence<Property>
      Properties;

  }; //module CompositeTypes

  interface RTIambassador
  {
    ////////////////////////////////////
    // Federation Management Services //
    ////////////////////////////////////

  //4.2
  void create_federation_execution (
     in wstring          federationExecutionName,
     in wstring          stringOfURLofFDD)
  raises (
     FederationExecutionAlreadyExists,
     CouldNotOpenFDD,
     ErrorReadingFDD,
     RTIinternalError);

     //4.3
     void destroy_federation_execution (
       in wstring federationExecutionName)
     raises (
       FederatesCurrentlyJoined,
       FederationExecutionDoesNotExist,
       RTIinternalError);

     //4.4
     FederateHandle join_federation_execution(
       in wstring                          federateType,
```

```
  in wstring                    federationExecutionName,
  in FederateAmbassador         federateReference)
raises (
  FederateAlreadyExecutionMember,
  FederationExecutionDoesNotExist,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

//4.5
void resign_federation_execution (
  in ResignActiontheResignAction)
raises (
  InvalidResignAction,
  OwnershipAcquisitionPending,
  FederateOwnsAttributes,
  FederateNotExecutionMember,
  RTIinternalError);

//4.6
void register_federation_synchronization_point (
  in wstring              synchronizationPointLabel,
  in UserSuppliedTag      theTag)
raises (
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

void register_federation_synchronization_point_with_set (
  in wstring              synchronizationPointLabel,
  in UserSuppliedTag      theTag,
  in CompositeTypes::FederateHandleSet synchronizationSet)
raises (
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

//4.9
void synchronization_point_achieved (
  in wstring        synchronizationPointLabel)
raises (
  SynchronizationPointLabelNotAnnounced,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 4.11
void request_federation_save (
```

```
  in wstring        label)
raises (
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

void request_federation_save_with_time (
  in wstring        label,
  in LogicalTime    theTime)
raises (
  LogicalTimeAlreadyPassed,
  InvalidLogicalTime,
  FederateUnableToUseTime,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 4.14
void federate_save_complete ()
raises (
  FederateHasNotBegunSave,
  FederateNotExecutionMember,
  RestoreInProgress,
  RTIinternalError);

void federate_save_not_complete ()
raises (
  FederateHasNotBegunSave,
  FederateNotExecutionMember,
  RestoreInProgress,
  RTIinternalError);

// 4.16
void query_federation_save_status ()
raises (
  FederateNotExecutionMember,
  SaveNotInProgress,
  RestoreInProgress,
  RTIinternalError);

// 4.18
void request_federation_restore (
  in wstring        label)
raises (
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);
```

```
// 4.22
void federate_restore_complete ()
raises (
  RestoreNotRequested,
  FederateNotExecutionMember,
  SaveInProgress,
  RTIinternalError);

void federate_restore_not_complete ()
raises (
  RestoreNotRequested,
  FederateNotExecutionMember,
  SaveInProgress,
  RTIinternalError);

// 4.24
void query_federation_restore_status ()
raises (
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreNotInProgress,
  RTIinternalError);

////////////////////////////////////////
// Declaration Management Services //
////////////////////////////////////////

// 5.2
void publish_object_class_attributes (
  in ObjectClassHandle  theClass,
  in CompositeTypes::AttributeHandleSet attributeList)
raises (
  ObjectClassNotDefined,
  AttributeNotDefined,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 5.3
void unpublish_object_class (
  in ObjectClassHandle theClass)
raises (
  ObjectClassNotDefined,
  OwnershipAcquisitionPending,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

void unpublish_object_class_attributes (
```

```
  in ObjectClassHandle              theClass,
  in CompositeTypes::AttributeHandleSet attributeList)
raises (
  ObjectClassNotDefined,
  AttributeNotDefined,
  OwnershipAcquisitionPending,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 5.4
void publish_interaction_class (
  in InteractionClassHandle theInteraction)
raises (
  InteractionClassNotDefined,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 5.5
void unpublish_interaction_class (
  in InteractionClassHandletheInteraction)
raises (
  InteractionClassNotDefined,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 5.6
void subscribe_object_class_attributes (
  in ObjectClassHandle       theClass,
  in CompositeTypes::AttributeHandleSet attributeList)
raises (
  ObjectClassNotDefined,
  AttributeNotDefined,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

void subscribe_object_class_attributes_passively (
  in ObjectClassHandle       theClass,
  in CompositeTypes::AttributeHandleSet attributeList)
raises (
  ObjectClassNotDefined,
  AttributeNotDefined,
  FederateNotExecutionMember,
  SaveInProgress,
```

```
  RestoreInProgress,
  RTIinternalError);

// 5.7
void unsubscribe_object_class (
  in ObjectClassHandle theClass)
raises (
  ObjectClassNotDefined,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

void unsubscribe_object_class_attributes (
  in ObjectClassHandle        theClass,
  in CompositeTypes::AttributeHandleSet attributeList)
raises (
  ObjectClassNotDefined,
  AttributeNotDefined,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 5.8
void subscribe_interaction_class (
  in InteractionClassHandle  theClass)
raises (
  InteractionClassNotDefined,
  FederateServiceInvocationsAreBeingReportedViaMOM,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

void subscribe_interaction_class_passively (
  in InteractionClassHandle  theClass)
raises (
  InteractionClassNotDefined,
  FederateServiceInvocationsAreBeingReportedViaMOM,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 5.9
void unsubscribe_interaction_class (
  in InteractionClassHandle  theClass)
raises (
  InteractionClassNotDefined,
  FederateNotExecutionMember,
```

```
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

////////////////////////////////
// Object Management Services //
////////////////////////////////

// 6.2
void reserve_object_instance_name (
  in wstring        theObjectName)
raises (
  IllegalName,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 6.4
ObjectInstanceHandle
register_object_instance (
  in ObjectClassHandle   theClass)
raises (
  ObjectClassNotDefined,
  ObjectClassNotPublished,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

ObjectInstanceHandle
register_object_instance_with_name (
  in ObjectClassHandle theClass,
  in wstring        theObjectName)
raises (
  ObjectClassNotDefined,
  ObjectClassNotPublished,
  ObjectInstanceNameNotReserved,
  ObjectInstanceNameInUse,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 6.6
void update_attribute_values (
  in ObjectInstanceHandle                                    theObject,
  in CompositeTypes::AttributeHandleValuePairSequence  theAttributes,
  in UserSuppliedTag                                         theTag)
raises (
  ObjectInstanceNotKnown,
```

```
  AttributeNotDefined,
  AttributeNotOwned,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

void
update_attribute_values_with_time (
  in  ObjectInstanceHandle                        theObject,
  in  CompositeTypes::AttributeHandleValuePairSequence  theAttributes,
  in  UserSuppliedTag                             theTag,
  in  LogicalTime                                 theTime,
  out MessageRetractionHandle                     handle,
  out boolean                                     retractionHandleIsValid)
raises (
  ObjectInstanceNotKnown,
  AttributeNotDefined,
  AttributeNotOwned,
  InvalidLogicalTime,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 6.8
void send_interaction (
  in InteractionClassHandle                       theInteraction,
  in CompositeTypes::ParameterHandleValuePairSequence theParameters,
  in UserSuppliedTag                              theTag)
raises (
  InteractionClassNotPublished,
  InteractionClassNotDefined,
  InteractionParameterNotDefined,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

void
send_interaction_with_time (
  in  InteractionClassHandle                      theInteraction,
  in  CompositeTypes::ParameterHandleValuePairSequence theParameters,
  in  UserSuppliedTag                             theTag,
  in  LogicalTime                                 theTime,
  out MessageRetractionHandle                     handle,
  out boolean                                     retractionHandleIsValid)
raises (
  InteractionClassNotPublished,
  InteractionClassNotDefined,
  InteractionParameterNotDefined,
```

```
  InvalidLogicalTime,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 6.10
void delete_object_instance (
  in ObjectInstanceHandle       objectHandle,
  in UserSuppliedTag            theTag)
raises (
  DeletePrivilegeNotHeld,
  ObjectInstanceNotKnown,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

void
delete_object_instance_with_time (
  in  ObjectInstanceHandle          objectHandle,
  in  UserSuppliedTag               theTag,
  in  LogicalTime                   theTime,
  out MessageRetractionHandle       handle,
  out boolean                       retractionHandleIsValid)
raises (
  DeletePrivilegeNotHeld,
  ObjectInstanceNotKnown,
  InvalidLogicalTime,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 6.12
void local_delete_object_instance (
  in ObjectInstanceHandle    objectHandle)
raises (
  ObjectInstanceNotKnown,
  FederateOwnsAttributes,
  OwnershipAcquisitionPending,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 6.13
void change_attribute_transportation_type (
  in ObjectInstanceHandle                 theObject,
  in CompositeTypes::AttributeHandleSet   theAttributes,
  in TransportationType                   theType)
```

```
raises (
  ObjectInstanceNotKnown,
  AttributeNotDefined,
  AttributeNotOwned,
  InvalidTransportationType,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 6.14
void change_interaction_transportation_type (
  in InteractionClassHandle   theClass,
  in TransportationType       theType)
raises (
  InteractionClassNotDefined,
  InteractionClassNotPublished,
  InvalidTransportationType,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 6.17
void request_attribute_value_update_for_instance (
  in ObjectInstanceHandle    theObject,
  in CompositeTypes::AttributeHandleSet theAttributes,
  in UserSuppliedTag         theTag)
raises (
  ObjectInstanceNotKnown,
  AttributeNotDefined,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

void request_attribute_value_update_for_class (
  in ObjectClassHandle                  theClass,
  in CompositeTypes::AttributeHandleSet theAttributes,
  in UserSuppliedTag                    theTag)
raises (
  ObjectClassNotDefined,
  AttributeNotDefined,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

////////////////////////////////////
// Ownership Management Services //
////////////////////////////////////
```

```
// 7.2
void unconditional_attribute_ownership_divestiture (
  in ObjectInstanceHandle                 theObject,
  in CompositeTypes::AttributeHandleSet   theAttributes)
raises (
  ObjectInstanceNotKnown,
  AttributeNotDefined,
  AttributeNotOwned,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 7.3
void negotiated_attribute_ownership_divestiture (
  in ObjectInstanceHandle                 theObject,
  in CompositeTypes::AttributeHandleSet   theAttributes,
  in UserSuppliedTag                      theTag)
raises (
  ObjectInstanceNotKnown,
  AttributeNotDefined,
  AttributeNotOwned,
  AttributeAlreadyBeingDivested,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 7.6
void confirm_divestiture (
  in ObjectInstanceHandle                 theObject,
  in CompositeTypes::AttributeHandleSet   theAttributes,
  in UserSuppliedTag                      theTag)
raises (
  ObjectInstanceNotKnown,
  AttributeNotDefined,
  AttributeNotOwned,
  AttributeDivestitureWasNotRequested,
 NoAcquisitionPending,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 7.8
void attribute_ownership_acquisition (
  in ObjectInstanceHandle                 theObject,
  in CompositeTypes::AttributeHandleSet   desiredAttributes,
  in UserSuppliedTag                      theTag)
raises (
```

```
    ObjectInstanceNotKnown,
    ObjectClassNotPublished,
    AttributeNotDefined,
    AttributeNotPublished,
    FederateOwnsAttributes,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError);

  // 7.9
  void attribute_ownership_acquisition_if_available (
    in ObjectInstanceHandle                 theObject,
    in CompositeTypes::AttributeHandleSet   desiredAttributes)
  raises (
    ObjectInstanceNotKnown,
    ObjectClassNotPublished,
    AttributeNotDefined,
    AttributeNotPublished,
    FederateOwnsAttributes,
    AttributeAlreadyBeingAcquired,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError);

  // 7.12
  CompositeTypes::AttributeHandleSet
  attribute_ownership_divestiture_if_wanted (
    in ObjectInstanceHandle                 theObject,
    in CompositeTypes::AttributeHandleSet   theAttributes)
  raises (
    ObjectInstanceNotKnown,
    AttributeNotDefined,
    AttributeNotOwned,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError);

  // 7.13
  void cancel_negotiated_attribute_ownership_divestiture (
    in ObjectInstanceHandle                 theObject,
    in CompositeTypes::AttributeHandleSet   theAttributes)
  raises (
    ObjectInstanceNotKnown,
    AttributeNotDefined,
    AttributeNotOwned,
    AttributeDivestitureWasNotRequested,
    FederateNotExecutionMember,
    SaveInProgress,
```

```
  RestoreInProgress,
  RTIinternalError);

// 7.14
void cancel_attribute_ownership_acquisition (
  in ObjectInstanceHandle                 theObject,
  in CompositeTypes::AttributeHandleSet   theAttributes)
raises (
  ObjectInstanceNotKnown,
  AttributeNotDefined,
  AttributeAlreadyOwned,
  AttributeAcquisitionWasNotRequested,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 7.16
void query_attribute_ownership (
  in ObjectInstanceHandle           theObject,
  in AttributeHandle                theAttribute)
raises (
  ObjectInstanceNotKnown,
  AttributeNotDefined,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 7.18
boolean
is_attribute_owned_by_federate (
  in ObjectInstanceHandle           theObject,
  in AttributeHandle                theAttribute)
raises (
  ObjectInstanceNotKnown,
  AttributeNotDefined,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

////////////////////////////////
// Time Management Services //
////////////////////////////////

// 8.2
void enable_time_regulation (
  in LogicalTimeInterval      theLookahead)
raises (
  TimeRegulationAlreadyEnabled,
```

```
  InvalidLookahead,
  InTimeAdvancingState,
  RequestForTimeRegulationPending,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 8.4
void disable_time_regulation ()
raises (
  TimeRegulationIsNotEnabled,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 8.5
void enable_time_constrained ()
raises (
  TimeConstrainedAlreadyEnabled,
  InTimeAdvancingState,
  RequestForTimeConstrainedPending,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 8.7
void disable_time_constrained ()
raises (
  TimeConstrainedIsNotEnabled,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 8.8
void time_advance_request (
  in LogicalTime              theTime)
raises (
  InvalidLogicalTime,
  LogicalTimeAlreadyPassed,
  InTimeAdvancingState,
  RequestForTimeRegulationPending,
  RequestForTimeConstrainedPending,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);
```

```
// 8.9
void time_advance_request_available (
  in LogicalTime              theTime)
raises (
  InvalidLogicalTime,
  LogicalTimeAlreadyPassed,
  InTimeAdvancingState,
  RequestForTimeRegulationPending,
  RequestForTimeConstrainedPending,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 8.10
void next_message_request (
  in LogicalTime        theTime)
raises (
  InvalidLogicalTime,
  LogicalTimeAlreadyPassed,
  InTimeAdvancingState,
  RequestForTimeRegulationPending,
  RequestForTimeConstrainedPending,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 8.11
void next_message_request_available (
  in LogicalTime        theTime)
raises (
  InvalidLogicalTime,
  LogicalTimeAlreadyPassed,
  InTimeAdvancingState,
  RequestForTimeRegulationPending,
  RequestForTimeConstrainedPending,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 8.12
void flush_queue_request (
  in LogicalTime        theTime)
raises (
  InvalidLogicalTime,
  LogicalTimeAlreadyPassed,
  InTimeAdvancingState,
  RequestForTimeRegulationPending,
  RequestForTimeConstrainedPending,
```

```
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 8.14
void enable_asynchronous_delivery()
raises (
  AsynchronousDeliveryAlreadyEnabled,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 8.15
void disable_asynchronous_delivery()
raises (
  AsynchronousDeliveryAlreadyDisabled,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 8.16
void query_GALT (
    out LogicalTime    time,
    out boolean        timeIsValid)
raises (
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 8.17
LogicalTime
query_logical_time ()
raises (
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 8.18
void query_LITS (
  out LogicalTime        time,
  out boolean            timeIsValid)
raises (
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);
```

```
// 8.19
void modify_lookahead (
  in LogicalTimeInterval  theLookahead)
raises (
  TimeRegulationIsNotEnabled,
  InvalidLookahead,
  InTimeAdvancingState,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 8.20
LogicalTimeInterval
query_lookahead ()
raises (
  TimeRegulationIsNotEnabled,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 8.21
void retract (
  in MessageRetractionHandle        theHandle)
raises (
  InvalidMessageRetractionHandle,
  TimeRegulationIsNotEnabled,
  MessageCanNoLongerBeRetracted,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 8.23
void change_attribute_order_type (
  in ObjectInstanceHandle                 theObject,
  in CompositeTypes::AttributeHandleSet   theAttributes,
  in OrderType                      theType)
raises (
  ObjectInstanceNotKnown,
  AttributeNotDefined,
  AttributeNotOwned,
  InvalidOrderType,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 8.24
```

```
void change_interaction_order_type (
  in InteractionClassHandle       theClass,
  in OrderType                    theType)
raises (
  InteractionClassNotDefined,
  InteractionClassNotPublished,
  FederateNotExecutionMember,
  InvalidOrderType,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

////////////////////////////////////
// Data Distribution Management //
////////////////////////////////////

// 9.2
RegionHandle
createRegion (in CompositeTypes::DimensionHandleSet dimensions)
raises (
  InvalidDimensionHandle,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 9.3
void commit_region_modifications (
  in CompositeTypes::RegionHandleSet regions)
raises (
  InvalidRegion,
  RegionNotCreatedByThisFederate,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 9.4
void delete_region (
  in RegionHandle         theRegion)
raises (
  InvalidRegion,
  RegionNotCreatedByThisFederate,
  RegionInUseForUpdateOrSubscription,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

//9.5
ObjectInstanceHandle
```

```
register_object_instance_with_regions (
  in ObjectClassHandle                                    theClass,
  in CompositeTypes::AttributeSetRegionSetPairSequence attributesAndRegions)
raises (
  ObjectClassNotDefined,
  ObjectClassNotPublished,
  AttributeNotDefined,
  AttributeNotPublished,
  InvalidRegion,
  RegionNotCreatedByThisFederate,
  InvalidRegionContext,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

ObjectInstanceHandle
register_object_instance_with_regions_with_name (
  in ObjectClassHandle                                    theClass,
  in CompositeTypes::AttributeSetRegionSetPairSequence attributesAndRegions,
  in wstring                                              theObject)
raises (
  ObjectClassNotDefined,
  ObjectClassNotPublished,
  AttributeNotDefined,
  AttributeNotPublished,
  InvalidRegion,
  RegionNotCreatedByThisFederate,
  InvalidRegionContext,
  ObjectInstanceNameNotReserved,
  ObjectInstanceNameInUse,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 9.6
void associate_regions_for_updates (
  in ObjectInstanceHandle                                 theObject,
  in CompositeTypes::AttributeSetRegionSetPairSequence attributesAndRegions)
raises (
  ObjectInstanceNotKnown,
  AttributeNotDefined,
  InvalidRegion,
  RegionNotCreatedByThisFederate,
  InvalidRegionContext,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);
```

```
// 9.7
void unassociate_regions_for_updates (
  in ObjectInstanceHandle                          theObject,
  in CompositeTypes::AttributeSetRegionSetPairSequence attributesAndRegions)
raises (
  ObjectInstanceNotKnown,
  AttributeNotDefined,
  InvalidRegion,
  RegionNotCreatedByThisFederate,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 9.8
void subscribe_object_class_attributes_with_regions (
  in ObjectClassHandle                             theClass,
  in CompositeTypes::AttributeSetRegionSetPairSequence attributesAndRegions)
raises (
  ObjectClassNotDefined,
  AttributeNotDefined,
  InvalidRegion,
  RegionNotCreatedByThisFederate,
  InvalidRegionContext,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

void subscribe_object_class_attributes_passively_with_regions (
  in ObjectClassHandle                             theClass,
  in CompositeTypes::AttributeSetRegionSetPairSequence attributesAndRegions)
raises (
  ObjectClassNotDefined,
  AttributeNotDefined,
  InvalidRegion,
  RegionNotCreatedByThisFederate,
  InvalidRegionContext,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 9.9
void unsubscribe_object_class_attributes_with_regions (
  in ObjectClassHandle                             theClass,
  in CompositeTypes::AttributeSetRegionSetPairSequence attributesAndRegions)
raises (
  ObjectClassNotDefined,
  AttributeNotDefined,
  InvalidRegion,
```

```
  RegionNotCreatedByThisFederate,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 9.10
void subscribe_interaction_class_with_regions (
  in InteractionClassHandle          theClass,
  in CompositeTypes::RegionHandleSet regions)
raises (
  InteractionClassNotDefined,
  InvalidRegion,
  RegionNotCreatedByThisFederate,
  InvalidRegionContext,
  FederateServiceInvocationsAreBeingReportedViaMOM,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

void subscribe_interaction_class_passively_with_regions (
  in InteractionClassHandle          theClass,
  in CompositeTypes::RegionHandleSet regions)
raises (
  InteractionClassNotDefined,
  InvalidRegion,
  RegionNotCreatedByThisFederate,
  InvalidRegionContext,
  FederateServiceInvocationsAreBeingReportedViaMOM,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 9.11
void unsubscribe_interaction_class_with_regions (
  in InteractionClassHandle          theClass,
  in CompositeTypes::RegionHandleSet regions)
raises (
  InteractionClassNotDefined,
  InvalidRegion,
  RegionNotCreatedByThisFederate,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

//9.12
void send_interaction_with_regions (
  in InteractionClassHandle                          theInteraction,
```

```
  in CompositeTypes::ParameterHandleValuePairSequence theParameters,
  in CompositeTypes::RegionHandleSet                  regions,
  in UserSuppliedTag                                  theTag)
raises (
  InteractionClassNotDefined,
  InteractionClassNotPublished,
  InteractionParameterNotDefined,
  InvalidRegion,
  RegionNotCreatedByThisFederate,
  InvalidRegionContext,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

void
send_interaction_with_regions_with_time (
  in  InteractionClassHandle                  theInteraction,
  in  CompositeTypes::ParameterHandleValuePairSequence theParameters,
  in  CompositeTypes::RegionHandleSet         regions,
  in  UserSuppliedTag                         theTag,
  in  LogicalTime                             theTime,
  out MessageRetractionHandle                 handle,
  out boolean                                 retractionHandleIsValid)
raises (
  InteractionClassNotDefined,
  InteractionClassNotPublished,
  InteractionParameterNotDefined,
  InvalidRegion,
  RegionNotCreatedByThisFederate,
  InvalidRegionContext,
  InvalidLogicalTime,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 9.13
void request_attribute_value_update_with_regions (
  in ObjectClassHandle                        theClass,
  in CompositeTypes::AttributeSetRegionSetPairSequence attributesAndRegions,
  in UserSuppliedTag                          theTag)
raises (
  ObjectClassNotDefined,
  AttributeNotDefined,
  InvalidRegion,
  RegionNotCreatedByThisFederate,
  InvalidRegionContext,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
```

```
  RTIinternalError);

//////////////////////////
// RTI Support Services //
//////////////////////////

// 10.2
ObjectClassHandle
get_object_class_handle (
  in wstring                theName)
raises (
  NameNotFound,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.3
wstring
get_object_class_name (
  in ObjectClassHandle  theHandle)
raises (
  InvalidObjectClassHandle,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.4
AttributeHandle
get_attribute_handle (
  in ObjectClassHandle      whichClass,
  in wstring                theName)
raises (
  InvalidObjectClassHandle,
  NameNotFound,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.5
wstring
get_attribute_name (
  in ObjectClassHandle      whichClass,
  in AttributeHandle        theHandle)
raises (
  InvalidObjectClassHandle,
  InvalidAttributeHandle,
  AttributeNotDefined,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.6
InteractionClassHandle
get_interaction_class_handle (
  in wstring                theName)
```

```
raises (
  NameNotFound,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.7
wstring
get_interaction_class_name (
  in InteractionClassHandle       theHandle)
raises (
  InvalidInteractionClassHandle,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.8
ParameterHandle
get_parameter_handle (
  in InteractionClassHandle       whichClass,
  in wstring                      theName)
raises (
  InvalidInteractionClassHandle,
  NameNotFound,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.9
wstring
get_parameter_name (
  in InteractionClassHandle       whichClass,
  in ParameterHandle              theHandle)
raises (
  InvalidInteractionClassHandle,
  InvalidParameterHandle,
  InteractionParameterNotDefined,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.10
ObjectInstanceHandle
get_object_instance_handle (
  in wstring                    theName)
raises (
  ObjectInstanceNotKnown,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.11
wstring
get_object_instance_name (
  in ObjectInstanceHandle    theHandle)
raises (
```

```
  ObjectInstanceNotKnown,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.12
DimensionHandle
get_dimension_handle (
  in wstring                theName)
raises (
  NameNotFound,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.13
wstring
get_dimension_name (
  in DimensionHandle        theHandle)
raises (
  InvalidDimensionHandle,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.14
long long
get_dimension_upper_bound (
  in DimensionHandle        theHandle)
raises (
  InvalidDimensionHandle,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.15
CompositeTypes::DimensionHandleSet
get_available_dimensions_for_class_attribute (
  in ObjectClassHandle      whichClass,
  in AttributeHandle        theHandle)
raises (
  InvalidObjectClassHandle,
  InvalidAttributeHandle,
  AttributeNotDefined,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.16
ObjectClassHandle
get_known_object_class_handle (
  in ObjectInstanceHandle   theObject)
raises (
  ObjectInstanceNotKnown,
  FederateNotExecutionMember,
  RTIinternalError);
```

```
// 10.17
CompositeTypes::DimensionHandleSet
get_available_dimensions_for_interaction_class (
  in InteractionClassHandle  theHandle)
raises (
  InvalidInteractionClassHandle,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.18
TransportationType
get_transportation_type (
  in wstring                 theName)
raises (
  InvalidTransportationName,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.19
wstring
get_transportation_name (
  in TransportationType      theType)
raises (
  InvalidTransportationType,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.20
OrderType
get_order_type (
  in wstring                 theName)
raises (
  InvalidOrderName,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.21
wstring
get_order_name (
  in OrderType               theType)
raises (
  InvalidOrderType,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.22
void enable_object_class_relevance_advisory_switch()
raises (
  FederateNotExecutionMember,
  ObjectClassRelevanceAdvisorySwitchIsOn,
```

```
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 10.23
void disable_object_class_relevance_advisory_switch()
raises (
  ObjectClassRelevanceAdvisorySwitchIsOff,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 10.24
void enable_attribute_relevance_advisory_switch()
raises (
  AttributeRelevanceAdvisorySwitchIsOn,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 10.25
void disable_attribute_relevance_advisory_switch()
raises (
  AttributeRelevanceAdvisorySwitchIsOff,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 10.26
void enable_attribute_scope_advisory_switch()
raises (
  AttributeScopeAdvisorySwitchIsOn,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 10.27
void disable_attribute_scope_advisory_switch()
raises (
  AttributeScopeAdvisorySwitchIsOff,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 10.28
void enable_interaction_relevance_advisory_switch()
```

```
raises (
  InteractionRelevanceAdvisorySwitchIsOn,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 10.29
void disable_interaction_relevance_advisory_switch()
raises (
  InteractionRelevanceAdvisorySwitchIsOff,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 10.30
CompositeTypes::DimensionHandleSet
get_dimension_handle_set(
  in RegionHandle region)
raises (
  InvalidRegion,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 10.31
CompositeTypes::RangeBounds
get_range_bounds(
  in RegionHandle          region,
  in DimensionHandle       dimension)
raises (
  InvalidRegion,
  RegionDoesNotContainSpecifiedDimension,
  FederateNotExecutionMember,
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 10.32
void set_range_bounds(
  in RegionHandle                region,
  in DimensionHandle             dimension,
  in CompositeTypes::RangeBounds bounds)
raises (
  InvalidRegion,
  RegionNotCreatedByThisFederate,
  RegionDoesNotContainSpecifiedDimension,
  InvalidRangeBound,
  FederateNotExecutionMember,
```

```
  SaveInProgress,
  RestoreInProgress,
  RTIinternalError);

// 10.33
long long
normalize_federate_handle(
  in FederateHandle                 theFederateHandle)
raises (
  InvalidFederateHandle,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.34
long long
normalize_service_group(
  in ServiceGroup                   group)
raises (
  InvalidServiceGroup,
  FederateNotExecutionMember,
  RTIinternalError);

// 10.35
CompositeTypes::Properties
initializeRTI(
  in CompositeTypes::Properties     theProperties)
raises (
  InitializePreviouslyInvoked,
  BadInitializationParameter,
  RTIinternalError);

// 10.36
void finalizeRTI()
raises (
  InitializeNeverInvoked,
  SomeFederateJoinedToAnExecution,
  RTIinternalError);

// 10.37
boolean
evoke_callback(
  in double seconds)
raises (
  FederateNotExecutionMember,
  RTIinternalError);

// 10.38
boolean
evoke_multiple_call_backs(
  in double minimumTime,
  in double maximumTime)
```

```
  raises (
    FederateNotExecutionMember,
    RTIinternalError);

  // 10.39
          void enable_callbacks()
  raises (
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError);

  // 10.40
  void disable_callbacks()
  raises (
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError);

  AttributeHandleFactory          get_attribute_handle_factory();
  DimensionHandleFactory          get_dimension_handle_factory();
  FederateHandleFactory           get_federate_handle_factory();
  InteractionClassHandleFactory   get_interaction_class_handle_factory();
  ObjectClassHandleFactory        get_object_class_handle_factory();
  ObjectInstanceHandleFactory     get_object_instance_handle_factory();
  ParameterHandleFactory          get_parameter_handle_factory();

  wstring getHLAversion();
};


interface FederateAmbassador {

  ////////////////////////////////////////
  // Federation Management Services ///
  ////////////////////////////////////////

  //4.7
  void synchronization_point_registration_succeeded(
    in wstring                              synchronizationPointLabel)
  raises (
    FederateInternalError);

  void synchronization_point_registration_failed(
    in wstring                              synchronizationPointLabel,
    in SynchronizationPointFailureReason reason)
  raises (
    FederateInternalError);
```

```
//4.8
void announce_synchronization_point(
  in wstring                    synchronizationPointLabel,
  in UserSuppliedTag            theTag)
raises (
  FederateInternalError);

//4.10
void federation_synchronized(
  in wstring                    synchronizationPointLabel)
raises (
  FederateInternalError);

//4.12
void initiate_federate_save(
  in wstring                    label)
raises (
  UnableToPerformSave,
  FederateInternalError);

void initiate_federate_save_with_time(
  in wstring                    label,
  in LogicalTime                time)
raises (
  InvalidLogicalTime,
  UnableToPerformSave,
  FederateInternalError);

// 4.15
void federation_saved ()
raises (
  FederateInternalError);

void federation_not_saved (
  in SaveFailureReason                  reason)
raises (
  FederateInternalError);

// 4.17
void federation_save_status_response (
  in CompositeTypes::SaveStatusSequence response)
raises (
  FederateInternalError);

// 4.19
void request_federation_restore_succeeded (
  in wstring                    label)
raises (
  FederateInternalError);

void request_federation_restore_failed (
```

```
  in wstring                              label)
raises (
  FederateInternalError);

// 4.20
void federation_restore_begun ()
raises (
  FederateInternalError);

// 4.21
void initiate_federate_restore (
  in wstring                              label,
  in FederateHandle                       theFederateHandle)
raises (
  SpecifiedSaveLabelDoesNotExist,
  CouldNotInitiateRestore,
  FederateInternalError);

// 4.23
void federation_restored ()
raises (
  FederateInternalError);

void federation_not_restored (
  in RestoreFailureReason                 reason)
raises (
  FederateInternalError);

// 4.25
void federation_restore_status_response (
  in CompositeTypes::RestoreStatusSequence response)
raises (
  FederateInternalError);


//////////////////////////////////////
// Declaration Management Services //
//////////////////////////////////////

// 5.10
void start_registration_for_object_class (
  in ObjectClassHandle                    theClass)
raises (
  ObjectClassNotPublished,
  FederateInternalError);

// 5.11
void stop_registration_for_object_class (
  in ObjectClassHandle                    theClass)
raises (
  ObjectClassNotPublished,
```

```
  FederateInternalError);

// 5.12
void turn_interactions_on (
  in InteractionClassHandle          theHandle)
raises (
  InteractionClassNotPublished,
  FederateInternalError);

// 5.13
void turn_interactions_off (
  in InteractionClassHandle          theHandle)
raises (
  InteractionClassNotPublished,
  FederateInternalError);

////////////////////////////////
// Object Management Services //
////////////////////////////////

// 6.3
void object_instance_name_reservation_succeeded (
  in wstring                              objectName)
raises (
  UnknownName,
  FederateInternalError);

void object_instance_name_reservation_failed (
  in wstring                              objectName)
raises (
  UnknownName,
  FederateInternalError);

// 6.5
void discover_object_instance (
  in ObjectInstanceHandle            theObject,
  in ObjectClassHandle               theObjectClass,
  in wstring                         objectName)
raises (
  CouldNotDiscover,
  ObjectClassNotRecognized,
  FederateInternalError);

// 6.7
void reflect_attribute_values (
  in ObjectInstanceHandle                                 theObject,
  in CompositeTypes::AttributeHandleValuePairSequence     theAttributes,
  in UserSuppliedTag                                      theTag,
  in OrderType                                            sentOrdering,
  in TransportationType                                   theTransport)
raises (
```

```
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  AttributeNotSubscribed,
  FederateInternalError);

void reflect_attribute_values_with_regions (
  in ObjectInstanceHandle                              theObject,
  in CompositeTypes::AttributeHandleValuePairSequence  theAttributes,
  in UserSuppliedTag                                   theTag,
  in OrderType                                         sentOrdering,
  in TransportationType                                theTransport,
  in CompositeTypes::RegionHandleSet                   sentRegions)
raises (
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  AttributeNotSubscribed,
  FederateInternalError);

void reflect_attribute_values_with_time (
  in ObjectInstanceHandle                              theObject,
  in CompositeTypes::AttributeHandleValuePairSequence  theAttributes,
  in UserSuppliedTag                                   theTag,
  in OrderType                                         sentOrdering,
  in TransportationType                                theTransport,
  in LogicalTime                                       theTime,
  in OrderType                                         receivedOrdering)
raises (
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  AttributeNotSubscribed,
  FederateInternalError);

void reflect_attribute_values_with_time_with_regions (
  in ObjectInstanceHandle                              theObject,
  in CompositeTypes::AttributeHandleValuePairSequence  theAttributes,
  in UserSuppliedTag                                   theTag,
  in OrderType                                         sentOrdering,
  in TransportationType                                theTransport,
  in LogicalTime                                       theTime,
  in OrderType                                   receivedOrdering,
  in CompositeTypes::RegionHandleSet             sentRegions)
raises (
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  AttributeNotSubscribed,
  FederateInternalError);

void reflect_attribute_values_with_retraction (
  in ObjectInstanceHandle                              theObject,
  in CompositeTypes::AttributeHandleValuePairSequence  theAttributes,
  in UserSuppliedTag                                   theTag,
```

```
    in OrderType                                    sentOrdering,
    in TransportationType                           theTransport,
    in LogicalTime                                  theTime,
    in OrderType                                    receivedOrdering,
    in MessageRetractionHandle                      retractionHandle)
  raises (
    ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    InvalidLogicalTime,
    FederateInternalError);

  void reflect_attribute_values_with_retraction_with_regions (
    in ObjectInstanceHandle                         theObject,
    in CompositeTypes::AttributeHandleValuePairSequence theAttributes,
    in UserSuppliedTag                              theTag,
    in OrderType                                    sentOrdering,
    in TransportationType                           theTransport,
    in LogicalTime                                  theTime,
    in OrderType                                    receivedOrdering,
    in MessageRetractionHandle                      retractionHandle,
    in CompositeTypes::RegionHandleSet              sentRegions)
  raises (
    ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    InvalidLogicalTime,
    FederateInternalError);

  // 6.9
  void receive_interaction (
    in InteractionClassHandle                       interactionClass,
    in CompositeTypes::ParameterHandleValuePairSequence theParameters,
    in UserSuppliedTag                              theTag,
    in OrderType                                    sentOrdering,
    in TransportationType                           theTransport)
  raises (
    InteractionClassNotRecognized,
    InteractionParameterNotRecognized,
    InteractionClassNotSubscribed,
    FederateInternalError);

  void receive_interaction_with_regions (
    in InteractionClassHandle                       interactionClass,
    in CompositeTypes::ParameterHandleValuePairSequence theParameters,
    in UserSuppliedTag                              theTag,
    in OrderType                                    sentOrdering,
    in TransportationType                           theTransport,
    in CompositeTypes::RegionHandleSet              sentRegions)
  raises (
    InteractionClassNotRecognized,
```

```
  InteractionParameterNotRecognized,
  InteractionClassNotSubscribed,
  FederateInternalError);

void receive_interaction_with_time (
  in InteractionClassHandle                          interactionClass,
  in CompositeTypes::ParameterHandleValuePairSequence theParameters,
  in UserSuppliedTag                                 theTag,
  in OrderType                                       sentOrdering,
  in TransportationType                              theTransport,
  in LogicalTime                                     theTime,
  in OrderType                                       receivedOrdering)
raises (
  InteractionClassNotRecognized,
  InteractionParameterNotRecognized,
  InteractionClassNotSubscribed,
  FederateInternalError);

void receive_interaction_with_time_with_regions (
  in InteractionClassHandle                          interactionClass,
  in CompositeTypes::ParameterHandleValuePairSequence theParameters,
  in UserSuppliedTag                                 theTag,
  in OrderType                                       sentOrdering,
  in TransportationType                              theTransport,
  in LogicalTime                                     theTime,
  in OrderType                                       receivedOrdering,
  in CompositeTypes::RegionHandleSet                 regions)
raises (
  InteractionClassNotRecognized,
  InteractionParameterNotRecognized,
  InteractionClassNotSubscribed,
  FederateInternalError);

void receive_interaction_with_retraction (
  in InteractionClassHandle                          interactionClass,
  in CompositeTypes::ParameterHandleValuePairSequence theParameters,
  in UserSuppliedTag                                 theTag,
  in OrderType                                       sentOrdering,
  in TransportationType                              theTransport,
  in LogicalTime                                     theTime,
  in OrderType                                       receivedOrdering,
  in MessageRetractionHandle                         theMessageRetractionHandle)
raises (
  InteractionClassNotRecognized,
  InteractionParameterNotRecognized,
  InteractionClassNotSubscribed,
  InvalidLogicalTime,
  FederateInternalError);

void receive_interaction_with_retraction_with_regions (
  in InteractionClassHandle                          interactionClass,
```

```
  in CompositeTypes::ParameterHandleValuePairSequence theParameters,
  in UserSuppliedTag                                  theTag,
  in OrderType                                        sentOrdering,
  in TransportationType                               theTransport,
  in LogicalTime                                      theTime,
  in OrderType                                        receivedOrdering,
  in MessageRetractionHandle                          theMessageRetractionHandle,
  in CompositeTypes::RegionHandleSet                  sentRegions)
raises (
  InteractionClassNotRecognized,
  InteractionParameterNotRecognized,
  InteractionClassNotSubscribed,
  InvalidLogicalTime,
  FederateInternalError);

// 6.11
void remove_object_instance (
  in ObjectInstanceHandle             theObject,
  in UserSuppliedTag                  theTag,
  in OrderType                        sentOrdering)
raises (
  ObjectInstanceNotKnown,
  FederateInternalError);

void remove_object_instance_with_time (
  in ObjectInstanceHandle             theObject,
  in UserSuppliedTag                  theTag,
  in OrderType                        sentOrdering,
  in LogicalTime                      theTime,
  in OrderType                        receivedOrdering)
raises (
  ObjectInstanceNotKnown,
  FederateInternalError);

void remove_object_instance_with_retraction (
  in ObjectInstanceHandle             theObject,
  in UserSuppliedTag                  theTag,
  in OrderType                        sentOrdering,
  in LogicalTime                      theTime,
  in OrderType                        receivedOrdering,
  in MessageRetractionHandle          retractionHandle)
raises (
  ObjectInstanceNotKnown,
  InvalidLogicalTime,
  FederateInternalError);

// 6.15
void attributes_in_scope (
  in ObjectInstanceHandle             theObject,
  in CompositeTypes::AttributeHandleSet theAttributes)
raises (
```

```
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  AttributeNotSubscribed,
  FederateInternalError);

// 6.16
void attributes_out_of_scope (
  in ObjectInstanceHandle                 theObject,
  in CompositeTypes::AttributeHandleSet   theAttributes)
raises (
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  AttributeNotSubscribed,
  FederateInternalError);

// 6.18
void provide_attribute_value_update (
  in ObjectInstanceHandle                 theObject,
  in CompositeTypes::AttributeHandleSet   theAttributes,
  in UserSuppliedTag                      theTag)
raises (
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  AttributeNotOwned,
  FederateInternalError);

// 6.19
void turn_updates_on_for_object_instance (
  in ObjectInstanceHandle                 theObject,
  in CompositeTypes::AttributeHandleSet   theAttributes)
raises (
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  AttributeNotOwned,
  FederateInternalError);

// 6.20
void turn_updates_off_for_object_instance (
  in ObjectInstanceHandle                 theObject,
  in CompositeTypes::AttributeHandleSet   theAttributes)
raises (
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  AttributeNotOwned,
  FederateInternalError);

////////////////////////////////////
// Ownership Management Services //
////////////////////////////////////

// 7.4
```

```
void request_attribute_ownership_assumption (
  in ObjectInstanceHandle                 theObject,
  in CompositeTypes::AttributeHandleSet   offeredAttributes,
  in UserSuppliedTag                      theTag)
raises (
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  AttributeAlreadyOwned,
  AttributeNotPublished,
  FederateInternalError);

// 7.5
void request_divestiture_confirmation (
  in ObjectInstanceHandle                 theObject,
  in CompositeTypes::AttributeHandleSet   offeredAttributes)
raises (
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  AttributeNotOwned,
  AttributeDivestitureWasNotRequested,
  FederateInternalError);

// 7.7
void attribute_ownership_acquisition_notification (
  in ObjectInstanceHandle                 theObject,
  in CompositeTypes::AttributeHandleSet   securedAttributes,
  in UserSuppliedTag                      theTag)
raises (
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  AttributeAcquisitionWasNotRequested,
  AttributeAlreadyOwned,
  AttributeNotPublished,
  FederateInternalError);

// 7.10
void attribute_ownership_unavailable (
  in ObjectInstanceHandle                 theObject,
  in CompositeTypes::AttributeHandleSet   theAttributes)
raises (
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  AttributeAlreadyOwned,
  AttributeAcquisitionWasNotRequested,
  FederateInternalError);

// 7.11
void request_attribute_ownership_release (
  in ObjectInstanceHandle                 theObject,
  in CompositeTypes::AttributeHandleSet   candidateAttributes,
  in UserSuppliedTag                      theTag)
```

```
raises (
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  AttributeNotOwned,
  FederateInternalError);

// 7.15
void confirm_attribute_ownership_acquisition_cancellation (
  in ObjectInstanceHandle                 theObject,
  in CompositeTypes::AttributeHandleSet   theAttributes)
raises (
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  AttributeAlreadyOwned,
  AttributeAcquisitionWasNotCanceled,
  FederateInternalError);

// 7.17
void inform_attribute_ownership (
  in ObjectInstanceHandle           theObject,
  in AttributeHandle                theAttribute,
  in FederateHandle                 theOwner)
raises (
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  FederateInternalError);

void attribute_is_not_owned (
  in ObjectInstanceHandle           theObject,
  in AttributeHandle                theAttribute)
raises (
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  FederateInternalError);

void attribute_is_owned_by_RTI (
  in ObjectInstanceHandle           theObject,
  in AttributeHandle                theAttribute)
raises (
  ObjectInstanceNotKnown,
  AttributeNotRecognized,
  FederateInternalError);

////////////////////////////////
// Time Management Services //
////////////////////////////////

// 8.3
void time_regulation_enabled (
  in LogicalTime time)
raises (
```

```
      InvalidLogicalTime,
      NoRequestToEnableTimeRegulationWasPending,
      FederateInternalError);

    // 8.6
    void time_constrained_enabled (
      in LogicalTime time)
    raises (
      InvalidLogicalTime,
      NoRequestToEnableTimeConstrainedWasPending,
      FederateInternalError);

    // 8.13
    void time_advance_grant (
      in LogicalTime theTime)
    raises (
      InvalidLogicalTime,
      JoinedFederateIsNotInTimeAdvancingState,
      FederateInternalError);

    // 8.22
    void request_retraction (
      in MessageRetractionHandle theHandle)
    raises (
      FederateInternalError);
  };
};
#endif
```

Other Specification Information *B*

B.1 Summary of optional versus mandatory interfaces

The specification proposes no optional interfaces.

B.2 Proposed compliance points

The specification proposes no separate compliance points.

B.3 Changes or extensions required to adopted OMG specifications

The specification proposes no changes or extensions to adopted OMG specifications.

Index

Index

Distributed Simulation Systems, v2.0
Reference Sheet

This is a revision to the formal DSS specification.

OMG documents used to create this version:

- Submission document: mfg/01-10-01
- FTF Report: dtc/02-05-01
- Convenience document: dtc/02-05-02