1516.1™

IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)— Federate Interface Specification

IEEE Computer Society

Sponsored by the
Simulation Interoperability Standards Organization/
Standards Activities Committee (SISO/SAC)

IEEE
3 Park Avenue
New York, NY 10016-5997, USA

18 August 2010

IEEE Std 1516.1™-2010
(Revision of
IEEE Std 1516.1-2000)

IEEE Std 1516.1™-2010
(Revision of
IEEE Std 1516.1-2000)

IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)— Federate Interface Specification

Sponsor

Simulation Interoperability Standards Organization/ Standards Activities Committee (SISO/SAC)
of the
IEEE Computer Society

Approved 25 March 2010

IEEE SA-Standards Board

Abstract: The High Level Architecture (HLA) has been developed to provide a common architecture for distributed modeling and simulation. The HLA defines an integrated approach that provides a common framework for the interconnection of interacting simulations. This document, the second in a family of three related HLA documents, defines the standard services of and interfaces to the HLA runtime infrastructure (RTI). These services are used by the interacting simulations to achieve a coordinated exchange of information when they participate in a distributed federation. The standards contained in this architecture are interrelated and need to be considered as a product set, when changes are made. They each have value independently.
Keywords: architecture, class attribute, data distribution management, federate, federation, federation execution, federation object model, HLA, instance attribute, instance attribute ownership, interaction class, object class, runtime infrastructure, simulation object model, time-constrained, time-regulating

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

PDF: ISBN 978-0-7381-6247-8 STD96059
Print: ISBN 978-0-7381-6248-5 STDPD96059

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information or soundness of any judgments contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied "**AS IS**."

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

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon his or her independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration. A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered the official position of IEEE or any of its committees and shall not be considered to be, nor be relied upon as, a formal interpretation of the IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Recommendations to change the status of a stabilized standard should include a rationale as to why a revision or withdrawal is required. Comments and recommendations on standards, and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854
USA

Introduction

This introduction is not part of IEEE Std 1516.1-2010, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Federate Interface Specification.

This document has been developed to record an international standard for the High Level Architecture (HLA). It serves as one of three related standards for the HLA. It defines the services and interfaces to be used by federates when participating in a federation execution.

This new version of the IEEE 1516 HLA was produced in 2004 through 2007 by the HLA Evolved Product Development Group of the Simulation Interoperability Standards Organization (SISO). It incorporates a number of updates based on practical application of earlier versions of the standard. The purpose of the new version is to better support development, deployment, and net-centricity of distributed simulations.

Major new additions include support for Web Services communication, modular information models [federation object models (FOMs) and simulation object models (SOMs)], improved extensible markup language (XML) features (XML Schemata as well as extensibility), improved fault tolerance support, support for update rate reduction, and dynamic link compatibility between different implementations.

Notice to users

Laws and regulations

Users of these documents should consult all applicable laws and regulations. Compliance with the provisions of this standard does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

This document is copyrighted by the IEEE. It is made available for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making this document available for use and adoption by public authorities and private users, the IEEE does not waive any rights in copyright to this document.

Updating of IEEE documents

Users of IEEE standards should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect. In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE Standards Association web site at http://ieeexplore.ieee.org/xpl/standards.jsp, or contact the IEEE at the address listed previously.

For more information about the IEEE Standards Association or the IEEE standards development process, visit the IEEE-SA web site at http://standards.ieee.org.

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: http://standards.ieee.org/reading/ieee/updates/errata/index.html. Users are encouraged to check this URL for errata periodically.

Interpretations

Current interpretations can be accessed at the following URL: http://standards.ieee.org/reading/ieee/interp/index.html.

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

Participants

At the time this standard was submitted to the IEEE-SA Standards Board for approval, the HLA Evolved Working Group had the following membership:

Roy Scrudder, *Chair*
Randy Saunders, *Vice Chair*
Björn Möller, *Vice Chair*
Katherine L. Morse, *Secretary*

Martin Adelantado
Bill Andrews
Fredrik Antelius
Joanne Atherton
Trevor Bakker
Shelby Barrett
Tolga Basturk
William Beavin
Emmet Beeker
Alan Berry
Mark Biwer
David Bodoh
Jake Borah
Steve Boswell
Derrick Briscoe
Dominique Canazzi
Andy Ceranowicz
Tram Chase
Scott Clarke
David Coppler
Anthony Cramp
Dannie Cutts
Timothy Daigle
Bradford Dillman
Steven Dix
Hoang Doan
Uwe Dobrindt
David Drake
David Edmondson
Craig Eidman
Gary Eiserman
Gary England
Jason Esteve
James Evans
Robert Farraher
John Fay
Reginald Ford
Masakazu Furuichi
Michael Gagliano
Ralph Gibson
Edward Gordon
Len Granowetter
Jean-Baptiste Guillerit
Paul Gustavson
Per Gustavsson
Steve Hall
Fawzi Hassaine
Mark Hazen
William Helfinstine
Amy Henninger
Frank Hill
Jim Hollenbach
Torbjörn Hultén
Jean-Louis Igarza
Mark S. Johnson
Stephen Jones
Stephen Jones
Gunnar Karlsson
Mikael Karlsson
Rosemarie Keener
James Kogler
Jonathan Labin
Jennifer Lewis
Mike Lightner
Reed Little
Laurie Litwin
Staffan Löf
Björn Löfstrand
Paul Lowe

Franklin Lue
Robert Lutz
Farid Mamaghani
Lee Marden
Kim Marshall
Regis Mauget
James McCall
Michael McGarity
Sandy McPherson
Rob Minson
Mike Montgomery
Neil Morris
William Oates
Gunnar Ohlund
Mike Papay
Trevor Pearce
Mikel Petty
Tim Pokorny
Edward Powell
Laurent Prignac
Guillaume Radde
Peter Ross
Chris Rouget
Joseph Sardella
Geoff Sauerborn
John Schloman
Kevin Seavey
Graham Shanks
Petr Shlyaev
John Shockley
Pierre Siron
Keith Snively
Susan Solick
Joseph Steel
Steffen Strassburger
Marcy Stutzman
Jerry Szulinski
Martin Tapp
Gary Thomas
Andreas Tolk
Cam Tran
Ben Watrous
Marc Williams
Annette Wilson
Douglas Wood
Roger Wuerfel
Troy Yee
William Zimmerman

The working group acknowledges the following Interface Specification Drafting Group members who also contributed to the preparation of this standard:

Reed Little, *Editor*

Steve Boswell
Dannie Cutts
Len Granowetter
Mikael Karlsson
Jonathan Labin
Mike Lightner
Robert Lutz
Björn Möller
Katherine Morse
Trevor Pearce
Chris Rouget
Graham Shanks
Keith Snively
Ben Watrous
Annette Wilson
Douglas Wood
Roger Wuerfel

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Ali Al Awazi
Bakul Banerjee
Steven Bezner
Jack Borah
Juan Carreon
Bertram Chase
Keith Chow
Tommy Cooper
Paul Croll
Dannie Cutts
Uwe Dobrindt
Sourav Dutta
Andre Fournier
Masakazu Furuichi
Len Granowetter
Randall Groves
Paul Gustavson
M. Hashmi
William Helfinstine
Rutger A. Heunks
Frank Hill
Werner Hoelzl
James Ivers
Mikael Karlsson
Mark Knight
James E. Kogler
Jonathan Labin
Susan Land
Reed Little
Björn Löfstrand
William Lumpkins
G. Luri
Robert Lutz
Edward McCall
James McCall
Gary Michel
William Milam
Björn Möller
Katherine Morse
Thomas Mullins
Michael S. Newman
Miroslav Pavlovic
T. Pearce
Ulrich Pohl
Jonathan Prescott
Jose Puthenkulam
Robert Robinson
Michael Rush
Peter Ryan
Randall Safier
Geoffrey Sauerborn
Randy Saunders
Bartien Sayogo
John Schloman
Roy Scrudder
Keith Snively
Susan Solick
Joseph Stanco
Thomas Starai
Gerald Stueve
Marcy Stutzman
Thomas Tullia
Cam Van Tran
Annette Wilson
Douglas Wood
Paul Work
Roger Wuerfel
Oren Yuen
Janusz Zalewski

When the IEEE-SA Standards Board approved this standard on 25 March 2010, it had the following membership:

Robert M. Grow, *Chair*
Richard H. Hulett, *Vice Chair*
Steve M. Mills, *Past Chair*
Judith Gorman, *Secretary*

Karen Bartleson
Victor Berman
Ted Burse
Clint Chaplin
Andy Drozd
Alexander Gelman
Jim Hughes
Young Kyun Kim
Joseph L. Koepfinger*
John Kulick
David J. Law
Hung Ling
Oleg Logvinov
Ted Olsen
Ronald C. Petersen
Thomas Prevost
Jon Walter Rosdahl
Sam Sciacca
Mike Seavey
Curtis Siller
Don Wright

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Satish Aggarwal, *NRC Representative*
Richard DeBlasio, *DOE Representative*
Michael Janezic, *NIST Representative*

Lisa Perry
IEEE Standards Program Manager, Document Development

Michael D. Kipness
IEEE Standards Program Manager, Technical Program Development

Contents

[a]All RTI-initiated services are denoted with a † (printer's dagger) after the service name.

IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Federate Interface Specification

1. Overview

1.1 Scope

This document defines the interface between federates (simulations, supporting utilities, or interfaces to live systems) and the underlying software services that support interfederate communication in a distributed simulation domain.

1.2 Purpose

The High Level Architecture (HLA) has been developed to provide a common architecture for distributed modeling and simulation (M&S). To facilitate interfederate communications, HLA federates interact with an underlying software infrastructure. This specification defines the standard services and interfaces to be used by the federates to support efficient information exchange when participating in a distributed federation execution. (A federation execution occurs when sets of federates are brought together to support an objective.)

1.3 Introduction

This document is being developed to define the HLA interface specification. The HLA requires a language-independent specification (LIS) and multiple language bindings to support intersimulation communication in a distributed simulation domain.

The formal definition of the M&S HLA comprises three main components: the HLA framework and rules specification (found in IEEE Std 1516™-2010[1]), the HLA federate interface specification (found in this document), and the HLA object model template (OMT) (found in IEEE Std 1516.2™-2010). This standard, IEEE Std 1516.1-2010, provides a complete description of the essential elements of the second component of the HLA, the interface specification.

The HLA is an integrated architecture that has been developed to provide a common architecture for M&S. The HLA requires that interfederate communications use a standard application programmer's interface (API). This specification defines the standard services and interfaces to be used by the federates in order to support efficient information exchange when participating in a distributed federation execution. Additionally, the capability for reuse of individual federates that adhere to these standard services and interfaces is increased.

This document provides a specification for the HLA functional interfaces between federates and the runtime infrastructure (RTI). The RTI provides services to federates in a way that is analogous to how a distributed operating system provides services to applications. These interfaces are arranged into seven basic service groups, and an eighth group of support services, as follows:

a) Federation management
b) Declaration management (DM)
c) Object management
d) Ownership management
e) Time management
f) Data distribution management (DMM)
g) Support services

The eight service groups describe the interface between the federates and the RTI and between the federates and the software services provided by the RTI for use by HLA federates. The RTI requires a set of services from the federate that are referred to as "RTI-initiated" and are denoted with a † (printer's dagger) after the service name.

1.4 Background

1.4.1 HLA federation object model (FOM) framework

A concise and rigorous description of the object model framework is essential to the specification of the interface between federates and the RTI and of the HLA services. The rules and terminology used to describe a FOM are described in IEEE Std 1516.2-2010. A simulation object model (SOM) describes salient characteristics of a federate to aid in its reuse and other activities focused on the details of its internal operation. As such, a SOM is not the concern of the RTI and its services. A FOM, on the other hand, deals with interfederate issues and is relevant to the use of the RTI. FOMs describe the set of object classes and interaction classes chosen for a planned federation along with the attributes of the object classes and parameters of the interaction classes.

[1]Information on references can be found in Clause 2.

Every HLA **object instance** is an instance of an object class found in the FOM. Object classes are chosen by the object model designer to facilitate an appropriate organizational scheme. Each object class has a set of attributes associated with it. An attribute is a distinct, identifiable portion of the object state. In this discussion, "attribute designator" refers to the attribute, and "attribute value" refers to its contents. From the federation perspective, the set of all attribute values for an object instance shall completely define the state of that instance. Federates may associate additional state information with an object instance that is not communicated between federates, but this action is outside the purview of the HLA FOM.

Federates use the state of the object instances as one of the primary means of communication. At any point during a federation execution, at most one federate shall be responsible for modeling a given object instance attribute. That federate provides instance attribute values to the other federates in the federation execution via HLA services. The federate providing the instance attribute values is said to be **updating** that instance attribute. Federates receiving those values are said to be **reflecting** that instance attribute.

The privilege to update a value for an instance attribute shall be uniquely held by a single joined federate at any given point during a federation execution. A joined federate that has the privilege to update values for an instance attribute is said to **own** that instance attribute. The RTI provides services that allow joined federates to exchange ownership of object instance attributes.

Each object instance shall have a designator. The value of an object instance designator shall be unique for each federation execution. Object instance designators shall be dynamically generated by the RTI.

The FOM framework also allows for interaction classes for each object model. The types of interactions possible and their parameters are specified within the FOM.

A **federation** is the combination of a particular FOM, a particular set of federates, and the HLA services. A federation is designed for a specific purpose using a commonly understood FOM and a set of federates that may associate their individual semantics with that object model. A federation execution is an instance of the *Create Federation Execution* service invocation and entails executing the federation with a specific FOM and RTI and using various execution details.

1.4.2 Relationship of HLA and object-oriented (OO) concepts

Although the HLA OMT is the standardized documentation structure for HLA object models, FOMs, FOM modules, SOMs, and SOM modules do not completely correspond to common definitions of object models in OO analysis and design (OOAD) techniques. In the OOAD literature, an object model is described as an abstraction of a system developed for the purpose of fully understanding the system. To achieve this understanding, most OO techniques recommend defining several views of the system. For HLA object models, the intended scope of the system description is much narrower and focuses specifically on requirements and capabilities for federate information exchange. For SOMs and SOM modules, the intent is to describe the public interface of the federate in terms of an identified set of supported HLA object classes and interaction classes. A more complete description of how a federate is designed and functions internally (for example, a traditional OO object model) should be provided via documentation resources other than the SOM. For FOMs and FOM modules, the intent is to describe information exchange that happens during a federation execution.

Differences between HLA and OOAD principles and concepts also appear at the individual object level. In the OOAD literature, objects are defined as software encapsulations of data and operations (methods). In the HLA, objects are defined entirely by the identifying characteristics (attributes), values of which are exchanged between federates during a federation execution. Any OO-related behaviors and operations that affect the values of HLA object attributes are kept resident in the federates. Although HLA class structures are driven by subscription requirements and FOM growth concerns, class structures in OO systems are typically driven by efficiency and maintainability concerns.

As discussed above, HLA object classes are described by the attributes that are defined for them. These are, in OO parlance, data members of the class. These attributes, which are abstract properties of HLA object classes, are referred to as class attributes. HLA object instances are spawned via an HLA service using an HLA object class as a template. Each attribute contained by an HLA object instance is called an instance attribute.

OO objects interact via message passing, in which one OO object invokes an operation provided by another OO object. HLA objects do not directly interact. It is the federates that interact, via HLA services, by updating instance attribute values or sending interactions. Also, responsibility for updating the instance attributes associated with an HLA object instance can be distributed among different federates in a federation (effectively distributing responsibility for maintaining the HLA object instance's state across the federation), whereas OO objects encapsulate state locally and associate update responsibilities with operations that are closely tied to the object's implementation in an OO programming language.

In addition to the stated semantic variations in shared terminology, other differences may also exist. Precise definitions of all HLA terms can be found in Clause 3.

1.4.3 General nomenclature and conventions

There are various entities (e.g., object classes, attributes, interaction classes, parameters, dimensions, regions, federates, object instances) referenced in this document that may have the following different "views":

— Name: textual or for communication between federates.
— Handle: capable of being manipulated by a computer or for communication between a federate and the RTI. A handle is originated by the RTI, federation execution-wide unique, and unpredictable. The handle for a class attribute that is inherited shall be the same as the handle that is assigned to the class attribute in the object class in which it is declared. The handle for a parameter that is inherited shall be the same as the handle that is assigned to the parameter in the interaction class in which it is declared.

The following types of names shall be case-sensitive:

— Federation execution
— Federate
— Object class
— Attribute
— Object instance
— Interaction class
— Parameter
— Order
— Transportation type
— Dimension

The arguments to the services described in this document will use different views of the entities depending on a particular RTI implementation. For clarity, the first 11 clauses of this document refer to only a generic view, known as a "designator," when referring to these entities, and the API annexes refer to the actual implementation-specific arguments. Clause 12 ties the two concepts together.

Some of the arguments to the services described in this document are in the form of a group that represents some association. These groupings can take several forms (depending on the requirements of the respective service), each of which is described below. The groupings may be implemented slightly differently in each API, but all implementations shall support the same semantics. These groupings are pairs, sets, collections, constrained sets of pairs, and collections of pairs. Collections are sets that permit duplication of elements.

They are used in cases in which duplication is unlikely and not really an error. As such, using a collection rather than a set retains the same effects, but without the expensive checks to eliminate duplicates that using a set would require. In these cases, it is preferred not to penalize the majority of users with the less efficient set just to catch errors that are anticipated to be infrequent.

The following data are needed for the implementation of a running RTI and federation executions:

— FOM Document Data (FDD): FOM information (e.g., class, attribute, parameter names) used by the RTI at runtime. Each federation execution needs FDD. FDD is provided to the RTI starting with the Management Object Model (MOM) and Initialization Module (MIM) and FOM modules specified in the *Create Federation Execution* service invocation. FDD will also be provided by the MIM that is automatically provided by the RTI or optionally provided in the *Create Federation Execution* service invocation. Additional FDD may be provided as joining federates specify additional FOM modules.

The following data may be needed for the implementation of a running RTI and federation executions:

— RTI initialization data (RID): RTI vendor-specific information needed to run an RTI. If required, RID are supplied when an RTI is initialized.

For all joined federate-initiated services in this specification (except 4.2, 4.5, 4.6, and 4.9), an implied supplied argument is a joined federate's conjointment with a federation execution. For all RTI-initiated services, an implied supplied argument is also a joined federate's connection to a federation execution. The manner in which these arguments are actually provided to the services is dependent on the RTI implementation and the particular API programming language; therefore, it is not shown in the service descriptions. Also, for the RTI-initiated services, some implicit preconditions are not stated explicitly because the RTI is assumed to be well behaved. For many RTI-initiated services, the "Federate is joined" precondition presumes the "Federate is connected" precondition.

As mentioned earlier, all RTI-initiated services are denoted with a † (printer's dagger) after the service name.

1.4.4 Organization of this document

The seven HLA service groups are specified in Clause 4 through Clause 10. Each service is described using several components:

— Name and description: service name and narrative describing the functionality of the service.
— Supplied arguments: arguments supplied to the service
— Returned arguments: arguments returned by the service.
— Preconditions: conditions that shall exist for the service to execute correctly.
— Postconditions: conditions that shall exist once the service has executed correctly.
— Exceptions: notifications of any irregularity that may occur during service invocation. Wherever possible (excepting situations like RTI internal error), exceptions are side-effect free. Exceptions are listed in order of increasing precedence. That is, if the conditions for more than one exception are met simultaneous, the RTI shall throw whichever of those exceptions comes later in the exception list.
— Reference state charts: reference to state chart(s) that are germane to the service.

After the clauses describing each of the service groups are a clause describing the MOM and then a clause that ties the service narratives with the APIs.

Annex A through Annex D contain HLA APIs for the following languages:

— Java
— C++
— Web Services Definition Language (WSDL)

Annex E contains additional rationale for elements in the main body of this standard.

Annex F contains the FDD XML Schema.

Annex G contains the MIM.

Annex H and Annex I contain a bibliography and list of figures, respectively.

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

IEEE Std 1516-2010, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Framework and Rules.[2]

IEEE Std 1516.2-2010, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Object Model Template (OMT) Specification.

The Unicode Consortium, Ed., *The Unicode Standard*, Version 3.0. Reading, MA: Addison-Wesley Developers Press, 2000.

[2]IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, Piscataway, NJ 08854, USA (http://standards.ieee.org/).

3. Definitions, abbreviations and acronyms

3.1 Definitions

For the purposes of this document, the following terms and definitions apply. *The IEEE Standards Dictionary: Glossary of Terms & Definitions* should be referenced for terms not defined in this clause.[3]

accuracy: The measure of the maximum deviation of an attribute or a parameter value in the simulation or federation from reality or some other chosen standard or referent.

active subscription: A request to the runtime infrastructure (RTI) for the kinds of data (class attributes as well as interactions) that the joined federate is currently interested in receiving. The RTI uses these data along with publication data received from other joined federates to support services, such as

a) *Start/Stop Registration*
b) *Turn On/Off Updates*
c) *Turn On/Off Interactions*

The RTI also uses the subscription (and publication) data to determine how to route data to the joined federates that require that data.

attribute: A named characteristic of an object class or object instance. *See also*: **class attribute**; **instance attribute**.

attribute ownership: The property of an instance attribute that gives a joined federate the capability to supply values for that instance attribute to its federation execution. *See also*: **instance attribute**.

available attributes: The set of declared attributes of an object class in union with the set of inherited attributes of that object class.

available dimensions:

a) Pertaining to an attribute: the dimensions associated with the class attribute in the federation object model (FOM). The available dimensions of an instance attribute are the available dimensions of the corresponding class attribute.
b) Pertaining to an interaction class: the dimensions associated with that interaction class in the FOM. The available dimensions of a sent interaction are the available dimensions of the interaction class specified in the *Send Interaction With Regions* service invocation.

NOTE—See 9.1.[4]

available parameters: The set of declared parameters of an interaction class in union with the set of inherited parameters of that interaction class.

base object model (BOM): A piece-part of a conceptual model, simulation object model (SOM), or federation object model (FOM) that can be used as a building block in the development and/or extension of a simulation or federation.

best effort: A transportation type that does not provide any guarantee regarding delivery (messages may be out of order, duplicate, or dropped).

[3] *The IEEE Standards Dictionary: Glossary of Terms & Definitions* is available at http://shop.ieee.org/.
[4] Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement the standard.

candidate discovery class: The registered class of an object instance, if subscribed. If the registered class of an object instance is not subscribed, the closest superclass of the registered class of the object instance to which the joined federate is subscribed. Candidate discovery class pertains to object instances only.

candidate received class: The sent class of an interaction, if subscribed. If the sent class of an interaction is not subscribed, the closest superclass of the sent class of the interaction to which the joined federate is subscribed. Candidate received class pertains to interactions only.

class: A description of a group of items with similar properties, common behavior, common relationships, and common semantics.

class attribute: A named characteristic of an object class denoted by a pair composed of an object class designator and an attribute designator.

class hierarchy: A specification of a class-subclass or "is-a" relationship between classes in a given domain.

coadunate: To attach an object instance handle and an object instance name to an object instance. The object instance name can be provided specifically by the federate or implicitly by the runtime infrastructure (RTI).

collection: A group of elements in which an element may occur multiple times (this definition corresponds to the mathematical notion of a bag).

collection of pairs: A group of pairs in which multiple pairs may have the same first component and a given pair may occur multiple times.

compliant object model: A High Level Architecture (HLA) federation object model (FOM) or simulation object model (SOM) that fully conforms with all of the rules and constraints specified in object model template (OMT).

constrained set of pairs: A group of pairs in which no two pairs have the same first component (this definition corresponds to the mathematical notion of a function). An example would be a group of instance attribute and value pairs; each instance attribute may have at most one value associated with it.

corresponding class attribute of an instance attribute: The class attribute that, from the perspective of a given joined federate, is the class attribute of the joined federate's known class for the object instance containing the instance attribute that has the same attribute designator as the instance attribute.

corresponding instance attributes of a class attribute: The instance attributes that, from the perspective of a given joined federate, are

a) Unowned instance attributes of object instances that have a known class at the joined federate equal to the object class of the class attribute and that have the same attribute designator as the class attribute, or

b) Instance attributes owned by the joined federate that belong to object instances that have a known class at the owning federate equal to the object class of the class attribute and that have the same attribute designator as the class attribute.

current federation object model (FOM): The union of the FOM modules and one Management Object Model (MOM) and Initialization Module (MIM) that have been specified in the creation of the federation execution or by any federate that has joined the federation execution. The sum operation is carried out according to the rules as prescribed in IEEE Std 1516.2-2010. When all FOM modules have been provided,

the current FOM will be equal to the FOM; and before this step has happened, the current FOM will be a true subset of the FOM.

current federation object model (FOM) Document Data (FDD): The data and information used to configure the federation execution that is the union of the FOM modules and one Management Object Model (MOM) and Initialization Module (MIM) that have been specified in the creation of the federation execution or by any federate that has joined the federation execution. The sum operation is carried out according to the rules as prescribed in IEEE Std 1516.2-2010. When all FOM modules have been provided, the current FDD will be equal to the FDD; and before this step has happened, the current FDD will be a true subset of the FDD.

datatype: A representation convention for a data element establishing its format, resolution, cardinality, and ordinality.

declared attributes: The set of class attributes of a particular object class that are listed in the federation object model (FOM) as being associated with that object class in the object class hierarchy tree.

declared parameters: The set of parameters of a particular interaction class that are listed in the federation object model (FOM) as being associated with that interaction class in the interaction class hierarchy tree.

default range: A range lower bound and a range upper bound, defined in the federation object model (FOM) Document Data (FDD) and specified in terms of [0, the dimension's upper bound), for a dimension.

default region: A multidimensional region provided by the runtime infrastructure (RTI) that is composed of one range for each dimension found in the federation object model (FOM) Document Data (FDD). The bounds of each of these ranges are [0, the range's dimension's upper bound). There is no way for a federate to refer to the default region.

NOTE—See 9.1.

delete: To invoke the *Delete Object Instance* service to eliminate a particular object instance. *See also*: **remove**.

NOTE—See 6.14.

designator: A generic view of arguments referenced in service descriptions. This view improves clarity in situations when arguments (mostly identifiers) to services have different views (implementations) due to a particular programming language or application programmer's interface (API).

dimension: A named interval. The interval is defined by an ordered pair of values, the first being the dimension lower bound and the second being the dimension upper bound. A runtime infrastructure (RTI) provides a predefined interval, whose lower and upper bounds are fixed as [0, the dimension's upper bound) as specified in the federation object model (FOM) Document Data (FDD). This interval provides a single basis for communication of all dimension-related data with an RTI. All normalized intervals communicated to the RTI will be subsets of this interval.

NOTE—See 9.1.

discover: To receive an invocation of the *Discover Object Instance †*[5] service for a particular object instance.

NOTE—See 6.9.

[5]All RTI-initiated services are denoted with a † (printer's dagger) after the service name.

discovered class: The class that was an object instance's candidate discovery class at a joined federate when that object instance was discovered by that joined federate. *See also*: **candidate discovery class.**

NOTE—See 5.1.2.

exception: Notification of any irregularity that may occur during a service invocation.

fault: An event that prevents the entire federation from executing in a High Level Architecture (HLA) compliant manner.

federate: An application that may be or is currently coupled with other software applications under a federation object model (FOM) Document Data (FDD) and a runtime infrastructure (RTI). *See also*: **federate application**; **joined federate.**

federate application: An application that supports the High Level Architecture (HLA) interface to a runtime infrastructure (RTI) and that is capable of joining a federation execution. A federate application may join the same federation execution multiple times or may join multiple federation executions. However, each time a federate application joins a federation execution, it is creating a new joined federate. *See also*: **joined federate**.

federation: A named set of federate applications and a common federation object model (FOM) that are used as a whole to achieve some specific objective.

federation execution: The actual operation, over time, of a set of joined federates that are interconnected by a runtime infrastructure (RTI).

federation objectives: The statement of the problem that is to be addressed by the establishment and execution of a federation.

federation object model (FOM): A specification defining the information exchanged at runtime to achieve a given set of federation objectives. This information includes object classes, object class attributes, interaction classes, interaction parameters, and other relevant information. The FOM is specified to the runtime infrastructure (RTI) using one or more FOM modules. The RTI assembles a FOM using these FOM modules and one Management Object Model (MOM) and Initialization Module (MIM), which is provided automatically by the RTI or, optionally, provided to the RTI when the federation execution is created.

federation object model (FOM) Document Data (FDD): The data and information in a FOM that are used by the *Create Federation Execution* service and successive *Join Federation Execution* service invocations to configure the federation execution.

federation object model (FOM) module: A partial FOM (containing some or all object model template (OMT) tables) that specifies a modular component of a FOM. A FOM module may contain classes not inherent to it but upon which the FOM module depends, i.e., superclasses to the modular components. These superclasses will be included in the FOM module as either complete or scaffolding definitions.

federation requirements: A statement that identifies a federation characteristic, constraint, process, or product that is unambiguous and testable and that is necessary for a federation to be acceptable for its intended use.

federation scenario: A set of initial conditions and timeline of significant events used within a federation execution to achieve federation objectives.

handle: An identifier originated/created by the runtime infrastructure (RTI) that is unique to a federation execution.

High Level Architecture (HLA) time axis: A totally ordered sequence of values in which each value typically represents an HLA instant of time in the physical system being modeled. For any two points, T1 and T2, on the time axis, if T1 < T2, T1 represents an instant of time that occurs before the instant represented by T2.

inherited attribute: A class attribute of an object class that was declared in a superclass of that object class in the object class hierarchy tree defined in the federation object model (FOM).

inherited parameter: An interaction parameter that was declared in a superclass of that interaction class in the interaction class hierarchy tree defined in the federation object model (FOM).

in scope: Of or pertaining to an instance attribute of an object instance for which all of the following apply:

a) The object instance is known to the joined federate.
b) The instance attribute is owned by another joined federate.
c) The instance attribute's corresponding class attribute is one of the following:
    1) A subscribed attribute of the known class of the object instance, or
    2) A subscribed attribute of the known class of the object instance with regions, and the update region set of the instance attribute at the owning joined federate 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.

    NOTE—See 6.1.

instance attribute: A named characteristic of an object instance denoted by a pair composed of the object instance designator and the attribute designator.

interaction: An explicit action taken by a federate that may have some effect or impact on another federate within a federation execution.

interaction class: A template for a set of characteristics that is common to a group of interactions. These characteristics correspond to the parameters that individual federates may associate with interactions.

interaction parameters: The information associated with an interaction that a federate potentially affected by the interaction may receive to calculate the effects of that interaction on its current state.

joined federate: A member of a federation execution, actualized by a federate application invoking the *Join Federation Execution* service as prescribed in IEEE Std 1516.1-2010. *See also*: **federate application**.

known class:

a) An object instance's registered class if the joined federate knows about the object instance as a result of having registered it, or
b) An object instance's discovered class if the joined federate knows about the object instance as a result of having discovered it.

known object instance: An object instance that a given joined federate has either registered or discovered and that the joined federate has not subsequently deleted (globally or locally) or been notified to remove. *See also*: **register**; **discover**; **delete**; **local delete**; **remove**.

last known good logical timestamp: The last timestamp to which a lost joined federate was successfully granted, as seen from the remaining High Level Architecture (HLA) compliant federation.

local delete: To invoke the *Local Delete Object Instance* service to inform the runtime infrastructure (RTI) that it is to treat the specified object instance as if the RTI had never notified the joined federate to discover the object instance (however, the object instance is not to be eliminated and may be rediscovered). *See also*: **delete**.

NOTE—See 6.16.

logical time: A federate's current point on the High Level Architecture (HLA) time axis. Federates making use of the time management services follow restrictions on what timestamps can be sent in timestamp order (TSO) messages (relative to their logical time) to ensure that federates receiving those messages receive them in TSO.

lookahead: A nonnegative value that establishes a lower value on the timestamps that can be sent in timestamp order (TSO) messages by time-regulating joined federates. Once established, a joined federate's lookahead value may be changed only by using the *Modify Lookahead* service. Each time-regulating joined federate must provide a lookahead value when becoming time-regulating.

lost joined federate: A previously joined federate that has been disconnected from the runtime infrastructure (RTI) as a result of a fault so that the joined federate can no longer continue in the federation execution in a High Level Architecture (HLA) compliant manner.

Management Object Model (MOM): A group of predefined High Level Architecture (HLA) constructs (object and interaction classes) that provide the following:

a) Access to federation execution operating information
b) Insight into the operations of joined federates and the runtime infrastructure (RTI)
c) Control of the functioning of the RTI, the federation execution, and the individual joined federates

Management Object Model (MOM) and Initialization Module (MIM): A subset of the federation object model (FOM) that contains object model template (OMT) tables that describe the High Level Architecture (HLA) MOM. The MIM also contains additional predefined HLA constructs such as object and interaction roots, datatypes, transportation types, and dimensions. HLA specifies a standard MIM that is incorporated into all FOM Document Data (FDD) automatically by the runtime infrastructure (RTI). The standard MIM can be replaced with a user-supplied MIM containing the standard MIM plus extensions.

message: A change of object instance attribute value(s), an interaction, or a deletion of an existing object instance, often associated with a particular point on the High Level Architecture (HLA) time axis, as denoted by the associated timestamp.

null designator: A designator reserved to indicate an empty value that is distinguishable from all other designators. A null designator is guaranteed to compare unequal to any other designator with a nonempty value.

object class: A fundamental element of a conceptual representation for a federate that reflects the real world at levels of abstraction and resolution appropriate for federate interoperability. An object class is a template for a set of characteristics that is common to a group of object instances. These characteristics correspond to the class attributes that individual federates may publish and to which other federates may subscribe.

object instance: A unique instantiation of an object class that is independent of all other instances of that object class. At any point during a federation execution, the state of a High Level Architecture (HLA) object instance is defined as the collection of the values of all its instance attributes.

object model: A system specification defined primarily by class characteristics and relationships. The High Level Architecture (HLA) idea of an object model is similar in many ways, but not identical, to the common idea of an object model in object-oriented literature.

object model framework: The rules and terminology used to describe High Level Architecture (HLA) object models.

order type: A runtime infrastructure (RTI) means of ordering messages originating from multiple joined federates that are delivered to a single joined federate. Different categories of service are defined with different characteristics regarding whether and how an RTI orders messages that are to be delivered to a joined federate.

out of scope: Of or pertaining to an instance attribute of an object instance for which one or more of the following are not true:

a) The object instance is known to the joined federate.
b) The instance attribute is owned by another joined federate.
c) The instance attribute's corresponding class attribute is one of the following:
    1) A subscribed attribute of the known class of the object instance, or
    2) A subscribed attribute of the known class of the object instance with regions, and the update region set of the instance attribute at the owning joined federate 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.

    NOTE—See 6.1.

overlap:

a) Pertaining to region sets: given two region sets, to have a region in each set such that the two regions overlap.
b) Pertaining to regions: given two regions, to have at least one dimension in common if and only if, for each dimension the regions have in common, their respective ranges in the common dimension overlap. If two regions do not have any dimensions in common, the regions do not overlap.
c) Pertaining to ranges: given two ranges (A = $[a_{lower}, a_{upper})$ and B = $[b_{lower}, b_{upper})$), to overlap if and only if either $a_{lower} = b_{lower}$ or ($a_{lower} < b_{upper}$ and $b_{lower} < a_{upper}$).

    NOTE—See 9.1.

owned: Pertaining to the relationship between an instance attribute and a joined federate: when the joined federate has the unique right to update the instance attribute's value.

owned instance attribute: An instance attribute that is explicitly modeled by the owning joined federate. A joined federate that owns an instance attribute has the unique responsibility to provide values for that instance attribute to the federation, through the runtime infrastructure (RTI), as documented in the federation object model (FOM) Document Data (FDD).

pair: A grouping of two related elements (a first component and a second component), the combination of which is treated as an entity. An example of a pair would be an instance attribute grouped with its current value.

parameter: A named characteristic of an interaction.

passel: A group of attribute handle/value pairs from an *Update Attribute Values* service invocation that are delivered together via a *Reflect Attribute Values †* service invocation. All pairs within the passel have the

same user-supplied tag, sent message order type, transportation type, receive message order type, timestamp (if present), and set of sent region designators (if present). A passel is a message.

passive subscription: A request to the runtime infrastructure (RTI) for the kinds of data (object classes and attributes as well as interactions) that the joined federate is currently interested in receiving. However, unlike an active subscription, this information is not used by the RTI to arrange for data to be delivered, nor is it used to tell publishing joined federates that another joined federate is subscribing to that data (by way of *Start/Stop Registration*, *Turn On/Off Updates,* or *Turn On/Off Interactions* service invocations). This form of subscription is provided to support certain types of logger joined federates.

promoted: Pertaining to an object instance, as known by a particular joined federate: having a discovered class that is a superclass of its registered class.

NOTE—See 5.1.3.

publish: To announce to a federation the information a federate may provide to the federation.

published:

a) Pertaining to an object class from the perspective of a given joined federate: having at least one available attribute of the object class that was an argument, along with the object class, to a *Publish Object Class Attributes* service invocation that was not subsequently unpublished via the *Unpublish Object Class Attributes* service.

   NOTE—See 5.1.2.

b) Pertaining to an interaction class from the perspective of a given joined federate: being an argument to a *Publish Interaction Class* service invocation that was not subsequently followed by an *Unpublish Interaction Class* service invocation for that interaction class.

   NOTE—See 5.1.3.

published attributes of an object class: The class attributes that have been arguments to *Publish Object Class Attributes* service invocations by a given joined federate for that object class that have not subsequently been unpublished (either individually or by unpublishing the whole object class), and possibly the **HLAprivilegeToDeleteObject** attribute for that object class.

NOTE—See 5.1.2.

range: A subset of a dimension, defined by an ordered pair of values, the first being the range lower bound and the second being the range upper bound. This pair of values defines a semi-open interval [range lower bound, range upper bound) (i.e., the range lower bound is the smallest member of the interval, and the range upper bound is just greater than any member of the interval).

NOTE—See 9.1.1.

range lower bound: The first component of the ordered pair of values that is part of a range.

NOTE—See 9.1.

range upper bound: The second component of the ordered pair of values that is part of a range.

NOTE—See 9.1.

received class: The class that was an interaction's candidate received class at the joined federate when that interaction was received at that joined federate via an invocation of the *Receive Interaction †* service.

NOTE—See 5.1.3.

received parameters: The set of parameters received when the *Receive Interaction †* service is invoked. These parameters consist of the subset of the sent parameters of an interaction that are available parameters of the interaction's received class.

NOTE—See 5.1.3.

receive order (RO): A characteristic of no ordering guarantee for messages. Messages that are received as RO messages will be received in an arbitrary order by the respective joined federate. A timestamp value will be provided with the message if one was specified when the message was sent, but that timestamp has no bearing on message receipt order.

reflect: To receive new values for one or more instance attributes via invocation of the *Reflect Attribute Values †* service.

NOTE—See 6.11.

reflected instance attribute: An instance attribute that is represented but not explicitly modeled in a joined federate. The reflecting joined federate accepts new values of the reflected instance attribute as they are produced by some other federation member and provided to it by the runtime infrastructure (RTI), via the *Reflect Attribute Values †* service.

region: A generic term that refers to either a region specification or a region realization. If not using data distribution management (DDM), region arguments may be ignored.

NOTE—See 9.1.

region realization: A region specification (set of ranges) that is associated with an instance attribute for update, with a sent interaction, or with a class attribute or interaction class for subscription. Region realizations are created from region specifications via the *Commit Region Modifications* (only when modifying a region specification from which region realizations are already derived), *Register Object Instance With Regions*, *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* services.

NOTE—See 9.1.1.

region specification: A set of ranges. Region specifications are created using the *Create Region* service, and a runtime infrastructure (RTI) is notified of changes to a region specification using the *Commit Region Modifications* service.

NOTE—See 9.1.1.

region template: A region template is an incomplete region specification where one or more dimensions have not been assigned ranges.

register: To invoke the *Register Object Instance* or the *Register Object Instance With Regions* service to create a unique object instance designator.

NOTE—See 6.8.

registered class: The object class that was an argument to the *Register Object Instance* or the *Register Object Instance With Regions* service invocation that resulted in the creation of the object instance designator for a given object instance.

regular interaction class description: A description of an interaction class that follows IEEE Std 1516.2-2010 and contains at a minimum the interaction class name and publish/subscribe indicator.

regular object class description: A description of an object class that follows IEEE Std 1516.2-2010 and contains at a minimum the object class name and publish/subscribe indicator.

remove: To receive an invocation of the *Remove Object Instance †* service for a particular object instance.

NOTE—See 6.15.

repeated description: A description of a construct (e.g., object class, interaction class, datatype, dimension) in a federation object model (FOM) module with a name and additional properties that are identical to a description that is already available in the current FOM.

resolution: The smallest resolvable value separating attribute or parameter values that can be discriminated. Resolution may vary with magnitude for certain datatypes.

retraction: An action performed by a federate to unschedule a previously scheduled message. Message retraction may be visible to the federate to whom the scheduled message was to be delivered. Retraction is widely used in classic event-oriented discrete event simulations to model behaviors such as preemption and interrupts.

runtime infrastructure (RTI): The software that provides common interface services during a High Level Architecture (HLA) federation execution for synchronization and data exchange.

runtime infrastructure (RTI) initialization data (RID): RTI vendor-specific information needed to run an RTI. If required, an RID is supplied when an RTI is initialized.

scaffolding interaction class description: A description of an interaction class in a federation object model (FOM) module or simulation object model (SOM) module that follows IEEE Std 1516.2-2010. However, it will contain only the name. The name of the scaffolding interaction class description is identical to the name of a regular interaction class description provided by another FOM/SOM module. Scaffolding interaction class descriptions act as placeholders in order to represent the class hierarchy from the other FOM/SOM module(s) from which a new regular interaction class description can be specified.

scaffolding object class description: A description of an object class in a federation object model (FOM) module or simulation object model (SOM) module that follows IEEE Std 1516.2-2010. However, it will contain only the name. The name of the scaffolding object class description is identical to the name of a regular object class description provided by another FOM/SOM module. Scaffolding object class descriptions act as placeholders in order to represent the class hierarchy from the other FOM/SOM module(s) from which a new regular object class description can be specified.

sent class: The interaction class that was an argument to the *Send Interaction* or *Send Interaction With Regions* service invocation that initiated the sending of a given interaction.

NOTE—See 5.1.3.

sent interaction: A specific interaction that is transmitted by a joined federate via the *Send Interaction* or *Send Interaction With Regions* service and received by other joined federates in the federation execution via the *Receive Interaction †* service.

sent parameters: The parameters that were arguments to the *Send Interaction* or *Send Interaction With Regions* service invocation for a given interaction.

NOTE—See 5.1.3.

set: A group of elements in which each element occurs at most once (this definition corresponds to the mathematical notion of sets). An example of a set would be a group of class attributes, each of which belongs to the same object class.

simulation object model (SOM): A specification of the types of information that an individual federate could provide to High Level Architecture (HLA) federations as well as the information that an individual federate can receive from other federates in HLA federations. The SOM is specified using one or more SOM modules. The standard format in which SOMs are expressed facilitates determination of the suitability of federates for participation in a federation.

simulation object model (SOM) module: A subset of the SOM that contains some or all object model template (OMT) tables. SOM modules and one optional Management Object Model (MOM) and Initialization Module (MIM) are used to specify the SOM of a federate. A SOM module contains complete or scaffolding definitions for all object classes and interaction classes that are superclasses of object classes and interaction classes in the same SOM module.

specified dimensions: The dimensions that are explicitly provided when the region specification is created or modified.

NOTE—See 9.1.2.

stop publishing: To take action that results in a class attribute that had been a published attribute of a class no longer being a published attribute of that class.

subclass: A class that adds additional detail to (specializes) another more generic class (superclass). A subclass, by inheriting the properties from its parent class (closest superclass), also inherits the properties of all superclasses of its parent as well.

subscribe: To announce to a federation the information a federate wants from the federation.

subscribed:

a) Pertaining to an object class from the perspective of a given joined federate: having subscribed attributes of that class or subscribed attributes of that class with regions, for some region. *See also*: **subscribed attributes of an object class**; **subscribed attributes of an object class with regions**.
b) Pertaining to an interaction class: being a subscribed interaction class or a subscribed interaction class with regions, for some region. *See also*: **subscribed interaction class**; **subscribed interaction class with regions**.

subscribed attributes of an object class: The class attributes that have been arguments to *Subscribe Object Class Attributes* service invocations by a given joined federate for a given object class that have not subsequently been unsubscribed, either individually or by unsubscribing the whole object class.

NOTE—See 5.1.2 and 5.6.

subscribed attributes of an object class with regions: The class attributes that have been arguments to *Subscribe Object Class Attributes With Regions* service invocations by a given joined federate for a given object class and a given region, assuming the joined federate did not subsequently invoke the *Unsubscribe Object Class Attributes With Regions* service for that object class and region.

NOTE—See 9.8.

subscribed interaction class: An interaction class that, from the perspective of a given joined federate, was an argument to a *Subscribe Interaction Class* or *Subscribe Interaction Class With Regions* service invocation that was not subsequently followed by an *Unsubscribe Interaction Class* or *Unsubscribe Interaction Class With Regions* service invocation for that interaction class.

NOTE—See 5.1.3 and 5.8.

subscribed interaction class with regions: An interaction class and a region that, from the perspective of a given joined federate, were arguments to a *Subscribe Interaction Class With Regions* service invocation that was not subsequently followed by an *Unsubscribe Interaction Class With Regions* service invocation for that interaction class and that region.

NOTE—See 9.10.

subscription region set: A set of regions used for subscription of a class attribute or used for subscription of an interaction class. *See also*: **used for subscription of a class attribute**; **used for subscription of an interaction class**.

superclass: A class that generalizes a set of properties that may be inherited by more refined (i.e., detailed) versions of the class. In High Level Architecture (HLA) applications, a class may have at most one immediate superclass (i.e., can inherit only from a single class at the next higher level of the class hierarchy).

synchronization point: A logical point in the sequence of a federation execution that all joined federates forming a synchronization set for that point attempt to reach and by which, if they are successful, they synchronize their respective processing.

Time Advancing state: The interval between a joined federate's request to advance its logical time and the corresponding grant by the runtime infrastructure (RTI). A joined federate may advance its logical time only by requesting a time advancement from the RTI via one of the following services:

a) *Time Advance Request*
b) *Time Advance Request Available*
c) *Next Message Request*
d) *Next Message Request Available*
e) *Flush Queue Request*

The joined federate's logical time will not actually be advanced until the RTI responds with a *Time Advance Grant †* service invocation at that joined federate.

time-constrained federate: A joined federate that may receive timestamp order (TSO) messages and whose time advances are constrained by other joined federates within a federation execution.

NOTE—See 8.1.

time management: A collection of High Level Architecture (HLA) services that support controlled message ordering and delivery to the cooperating joined federates within a federation execution in a way that is consistent with federation requirements.

time-regulating federate: A joined federate that may send timestamp order (TSO) messages and that constrains the time advances of other joined federates within a federation execution.

NOTE—See 8.1.

timestamp (of message or save): The value of the timestamp argument provided to the relevant service invocation.

timestamp order (TSO): An ordering of messages provided by a runtime infrastructure (RTI) for joined federates making use of time management services and messages containing timestamps. Messages having different timestamps are said to be delivered in TSO if for any two messages, M1 and M2 (timestamped with T1 and T2, respectively), that are delivered to a single joined federate where T1 < T2, then M1 is delivered before M2. Messages having the same timestamp will be delivered in an arbitrary order (i.e., no tie-breaking mechanism is provided by an RTI).

transportation type: A runtime infrastructure (RTI) provided means of transmitting messages between joined federates. Different categories of service are defined with different characteristics such as reliability of delivery and message latency.

unspecified dimensions: The available dimensions of a class attribute, instance attribute, interaction class, or sent interaction less the specified dimensions of the region specification from which the region realization is derived.

NOTE—See 9.1.3.

update: To invoke the *Update Attribute Values* service for one or more instance attributes.

NOTE—See 6.10.

update rate: The rate at which instance attribute values are provided, either by an owning joined federate to the runtime infrastructure (RTI) or by the RTI to a subscribing joined federate.

update region set: A set of regions used for sending interactions or used for update of instance attributes. *See also*: **used for sending**; **used for update**.

used for sending:

a) Pertaining to a region and the specified interaction class designator: being an argument in the *Send Interaction With Regions* service.

b) Pertaining to the default region: when the specified interaction class designator is an argument in the *Send Interaction* service.

NOTE—See 9.1.

used for subscription of a class attribute:

a) Pertaining to a region, an object class, and a class attribute: when the class attribute is a subscribed attribute of the object class with that region. *See also*: **subscribed attributes of an object class with regions.**

   NOTE—See 9.1.

b) Pertaining to the default region: when the specified class attribute is a subscribed attribute of the specified class. *See also*: **subscribed attributes of an object class with regions.**

   NOTE—See 9.1.

used for subscription of an interaction class:

a) Pertaining to a region and an interaction class: when the interaction class is a subscribed interaction class with regions. *See also*: **subscribed interaction class**.

   NOTE—See 9.1.

b) Pertaining to the default region: when the specified interaction class is a subscribed interaction class. *See also*: **subscribed interaction class**.

NOTE—See 9.1.

used for update:

a) Pertaining to a region that, along with the specified object instance and attribute designators, has been used as an argument in either the *Register Object Instance With Regions* service or the *Associate Regions For Updates* service: when the region has not subsequently been used along with the specified object instance designator as an argument in the *Unassociate Regions For Updates* service, nor has the joined federate subsequently lost ownership of the specified instance attribute(s).

NOTE—See 9.1.

b) Pertaining to the default region: when the specified instance attribute(s) is not currently used for update with any other region.

NOTE—See 9.1.

valid federate designator: A federate designator which, during the federation execution, was returned by the *Join Federation Execution* service to some joined federate. The federate does not have to remain a joined federate for its federate designator to remain a valid federate designator.

wall-clock time: Time as determined by a chronometer such as a wristwatch or wall clock.

3.2 Abbreviations and acronyms

The following abbreviations and acronyms pertain to IEEE Std 1516-2010, this standard, and IEEE Std 1516.2-2010.

ADT	abstract datatype
API	application programmer’s interface
BNF	Backus Naur Form
BOM	base object model
DDM	data distribution management
DIF	data interchange format
DM	declaration management
DTD	document type declaration
FDD	FOM Document Data
FEDEP	Federation Development and Execution Process
FOM	federation object model
GALT	Greatest Available Logical Time
HLA	High Level Architecture
JAR	Java Archive
LIS	language-independent specification
LITS	Least Incoming Timestamp
LRC	Local Runtime Component
MIM	MOM and Initialization Module

MOM	Management Object Model
M&S	modeling and simulation
NA	not applicable
OMT	object model template
OO	object oriented
OOAD	object-oriented analysis and design
POC	point of contact
RID	RTI initialization data
RO	receive order
RTI	runtime infrastructure
SOM	simulation object model
TRADT	time representation abstract datatype
TSO	timestamp order
WSDL	Web Services Definition Language
WSPRC	Web Service Provider RTI Component
XML	extensible markup language

4. Federation management

4.1 Overview

Federation management refers to the creation, dynamic control, modification, and deletion of a federation execution.

Before any interaction can take place between a federate and an RTI, such as creating a federation execution or joining, the unjoined federate shall perform a *Connect* to the RTI. When a federate has resigned and has no further intent to join or create federation executions, it shall perform a *Disconnect*.

The exact manner in which RTI-initiated services (callbacks) are invoked on a federate by the RTI is programming-language specific. However, the HLA defines two callback models that govern when callbacks may be invoked. The two callback models are: IMMEDIATE and EVOKED. In the IMMEDIATE model, the RTI shall invoke callbacks immediately when they are available, unless the callbacks have been disabled using the *Disable Callbacks* service. In the EVOKED model, callbacks are only made when the federate calls the *Evoke Callback/Evoke Multiple Callbacks* services, unless callbacks have been disabled using the *Disable Callbacks* service. RTIs may implement both callback models, but shall implement at least one. The callback model is specified as an argument to the *Connect* service.

Figure 1 shows the overall state of a federation execution as certain basic federation management services are employed. The figure starts in the Federation Execution Does Not Exist state, representing that the RTI has been started up, but that no services have been invoked relative to the federation execution. After the federation execution is created, it is now in the No Joined Federates state. This state represents the clean, or freshly initialized, federation execution; no federates are joined, no object instances exist, and no messages are queued. From the point that the first federate joins this execution until the point that the last federate resigns from the execution, the federation execution is in the Supporting Joined Federates state. This state represents an operational federation execution in which the services documented in this specification are usable. If all joined federates resign before a willing joined federate is identified, all unowned instance attributes shall cease to exist as if their object instances were deleted at the time of the last federate resign.

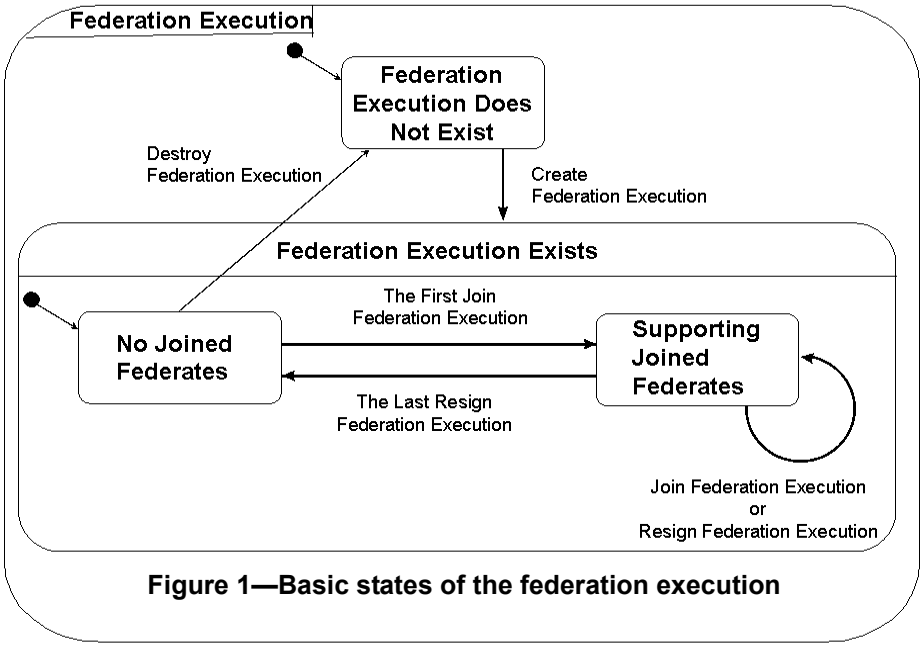

Figure 1—Basic states of the federation execution

Once a federation execution exists, federates may join and resign from it in any sequence that is meaningful to the federation user. Figure 2 presents a generalized view of the basic relationship between a federate and the RTI during the federate participation in a federation execution. The broad arrows in Figure 2 represent the general invocation of HLA service groups and are not intended to demonstrate strict ordering requirements on the use of the services. An RTI shall allow an application to participate in a federation as multiple joined federates, and an RTI shall allow a single application to participate in multiple federations as a joined federate or federates.

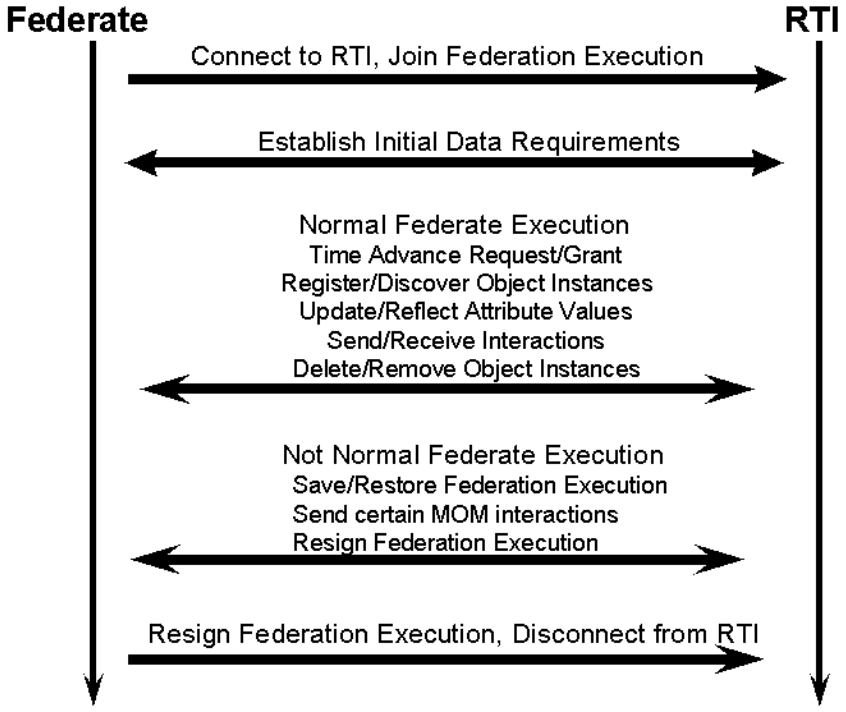

Figure 2—Overall view of federate-to-RTI relationship

The state diagram in Figure 3 is the first of a series of hierarchical state diagrams that formally describe the exchanges between federates and an RTI. These state diagrams are formal, accurate descriptions of federate state information depicted in the highly structured, compact, and expressive state chart notation pioneered by David Harel [B2]. This specification contains two independent sets of state charts. The first set, beginning in Figure 3 and including all other state charts except Figure 6 through Figure 9, describes the state of a given federate, from the perspective of that federate, in varying levels of detail. The second set, Figure 6 through Figure 9, describes the state of a federation execution (including the RTI and all joined federates) strictly with respect to the use of synchronization points. Finally, Figure 1 is unrelated to either of these sets of state charts. It is a stand-alone overview of the basic states of a federation execution.

The next few subclauses describe the first two of these state charts in detail as a way of introducing some of Harel's notation and providing an understanding of how the complete collection of state charts in this document are hierarchically interrelated.

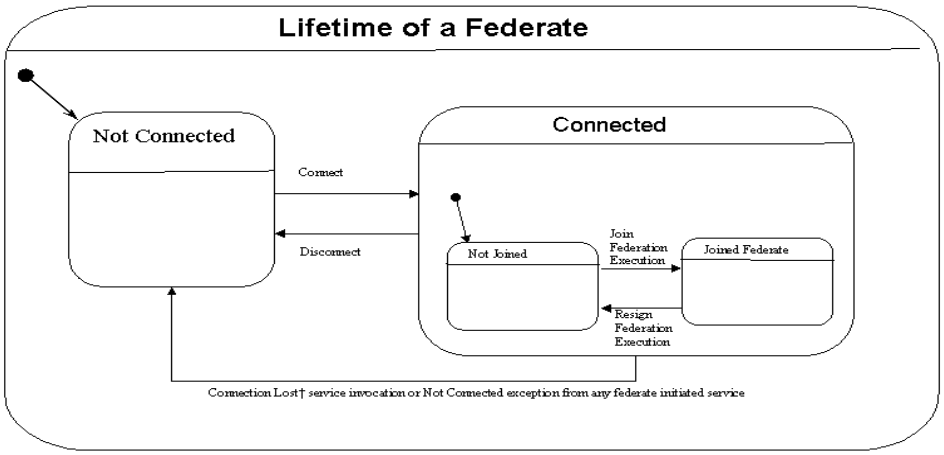

Figure 3—Lifetime of a federate

4.1.1 Federate lifetime

As shown in Figure 3, the initial state of any federate is "Not Connected." Before a federate can work with an RTI, such as to create a federation execution or join one, the federate shall perform a *Connect* to the RTI. The *Connect* service allows the RTI to set up and initialize its communication infrastructure. After successful invocation of the *Join Federation Execution* service, a federate will be in the Joined Federate state, where distributed simulation activities take place. If an infrastructure problem is detected during these activities, the *Connection Lost †*[5] service notifies the federate that a fault has been detected and that it has been forced to the Not Connected state. If previously joined, it shall be now no longer joined to the federation execution. As described in 4.1.5, the effect will be as if the federate had resigned with a resign action specified by the Automatic Resign Directive. Following successful simulation activities, the federate invokes the *Resign Federation Execution* service to leave the Joined Federate state. When a federate has resigned and has no further intent to join or create federation executions, it shall perform a *Disconnect*. The *Disconnect* service allows the RTI to break down and clean up its communication infrastructure. Figure 4 shows the federate's lifetime in the Joined Federate state.

As indicated by the dashed line in Figure 4, the Joined Federate state consists of two parallel state machines: one having to do with whether the joined federate is in the process of saving or restoring the Joined Federate state (depicted to the left of the dashed line), and the other having to do with whether the joined federate is permitted to perform normal activity (depicted to the right of the dashed line). While in the Joined Federate state, the joined federate is simultaneously in both a state depicted in the state machine to the left of the dashed line and a state depicted in the state machine to the right of the dashed line. Initially, upon entering the Joined Federate state, the joined federate will be in the Active and Normal Activity Permitted states, as indicated by the dark-circle start transitions. There are interdependencies between these two parallel state machines and between the state machine on the left and the Temporal state machine, Figure 17 (in 8.1.7). These interdependencies are depicted by the guards (shown within square brackets) that are associated with some state transitions. If a transition has a guard associated with it, then when the assertion within the guard is true, the joined federate may make the associated transition from one state to another. If a transition has only a guard (with no associated service invocation), the joined federate will immediately make the associated transition as soon as the guard is true. As an example of an interdependency between the two parallel state machines depicted in the Joined Federate state, if a joined federate that is in the Active state receives a *Federation Restore Begun †* service invocation, it will transition into the Prepared to Restore state (as indicated by the label on the transition from the Active state to the Prepared to Restore state). Once the

[5]All RTI-initiated services are denoted with a † (printer's dagger) after the service name.

joined federate enters the Prepared to Restore state, it also enters the Normal Activity Not Permitted state (as indicated by the guard on the transition from the Normal Activity Permitted state to the Normal Activity Not Permitted state). In other words, the guards impose the following constraints on a joined federate: A joined federate may be in the Normal Activity Permitted state (right side) if and only if it is also in the Active Federate state (left side); and a joined federate may be in the Normal Activity Not Permitted state (right side) if and only if it is also in the Instructed to Save, Saving, Waiting for Federation to Save, Prepared to Restore, Restoring, Waiting for Federation to Restore, or Waiting for Restore to Begin state (left side).

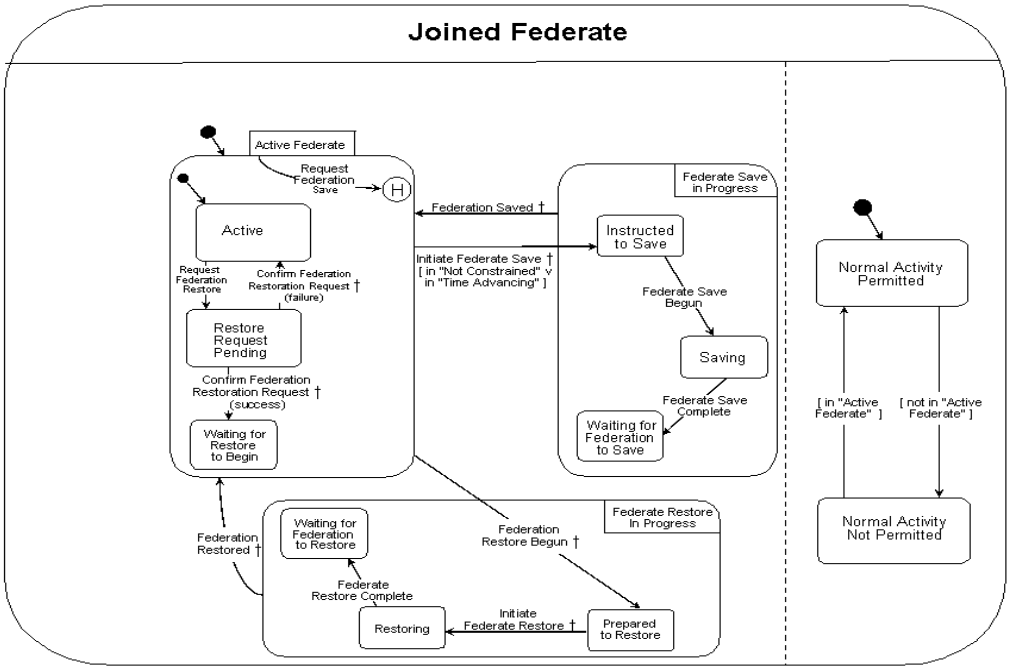

Figure 4—Joined Federate state

The interdependency between the state machine on the left and the Temporal state machine, Figure 17 (in 8.1.7), is this: a joined federate that is in the Active state will not receive an invocation of the *Initiate Federate Save †* service unless that joined federate is either in the Not Constrained state or the Time Advancing state. The Not Constrained state and Time Advancing state are depicted in Figure 17. The fact that these two states related to time management are mentioned in the guard on the transition from the Active state to the Instructed to Save state demonstrates the interdependencies between a joined federate's save/restore state and its Temporal state. Specifically, it indicates that a joined federate shall either not be constrained by time management or be in a position to receive a time advance grant in order for it to receive an invocation of the *Initiate Federate Save †* service.

A state chart notation that needs additional explanation is the shallow history state, depicted by a state labeled **H.** This notation means to enter the most recently visited of all the states at the level of the **H** state or enter the initial state (as indicated by the dark-circle start transition) if this occurrence is the first entry. For example, in Figure 4, there is a shallow history state within the Active Federate state. The transition for the *Request Federation Save* service originates from the outer edge of the Active Federate state, i.e., the transition can occur from any state within the Active Federate state (i.e., Active or Restore Request Pending). The transition terminates at the shallow history state; in other words, on completing the transition, the federate will return to the state within the Active Federate state in which it was previously (i.e., it will return to Active or to Restore Request Pending). Transition from a superstate to a shallow history state is a

common idiom denoting that the transition can occur while in any substate of the superstate and will result in the machine's returning to the same state in which it was prior to the transition.

If a joined federate is in the Normal Activity Permitted state, the joined federate may perform normal joined federate activity, such as registering and discovering object instances, publishing and subscribing to object class attributes and interactions, updating and reflecting instance attribute values, sending and receiving interactions, deleting and removing object instances, and requesting or receiving time advance grants. During federation save or restore, a joined federate will enter the Normal Activity Not Permitted state. In this state, the only permitted activities are those performed by the joined federate and with the joined federate (by the RTI) related to the save or restore in progress. Additionally, while in the Normal Activity Not Permitted state, a joined federate may

— Resign the federation execution or
— Send certain save- or restore-related interactions.

The Normal Activity Permitted state, simple as it may appear in Figure 4, actually contains all of the other states that appear in the first set of state charts that appear subsequently in this document. Together, these state charts formally describe the state of a joined federate from that joined federate's perspective. These state charts are complete in the sense that all transitions shown represent legal operations and transitions that are not shown represent illegal operations. The RTI shall generate exceptions if illegal operations are invoked. The state charts do not include MOM or data distribution management (DDM) activities.

The Normal Activity Permitted state depicted in Figure 4 is enhanced in Figure 5 to identify the major portions of the Joined Federate state. Additionally, the states in Figure 5 are further detailed as follows:

— Temporal state (Figure 17, in 8.1.7)
— States associated with each object class (Figure 10, in 5.1.4)
— States associated with each interaction class (Figure 13, in 5.1.4)

When a federate enters the Joined Federate state, the joined federate will have a Temporal state, object class states, and interaction class states. The joined federate will have an Object Class state for each object class that is defined in the FDD that is associated with the federation execution. Likewise, the joined federate will have an Interaction Class state for each interaction class that is defined in the FDD. A joined federate will be in the Temporal state, in each of these Object Class states, and in each of these Interaction Class states simultaneously (as depicted by the dashed lines separating the state machines within the Normal Activity Permitted state). Time management is detailed in Figure 17. The state of an arbitrary object class is detailed in Figure 10, and the state of an arbitrary interaction class is detailed in Figure 13. Indices, such as "i" in Object Class (i), are used in the names of many state charts to denote that the state chart refers to an arbitrary occurrence of something. These indices are used consistently throughout this specification and are described as follows:

i Object class designator
j Attribute designator
k Object instance designator
m Interaction class designator
L Synchronization label
s *s*th joined federate; also used as F_s

Another state chart notation that needs additional explanation is the deep history state, depicted by a state labeled $\mathbf{H}^*$**.** This notation means to enter the most recently visited of all the states at the level of the $\mathbf{H}^*$ state or enter the initial state (as indicated by the dark-circle start transition) if this occurrence is the first entry and to do the same for all relevant substates.

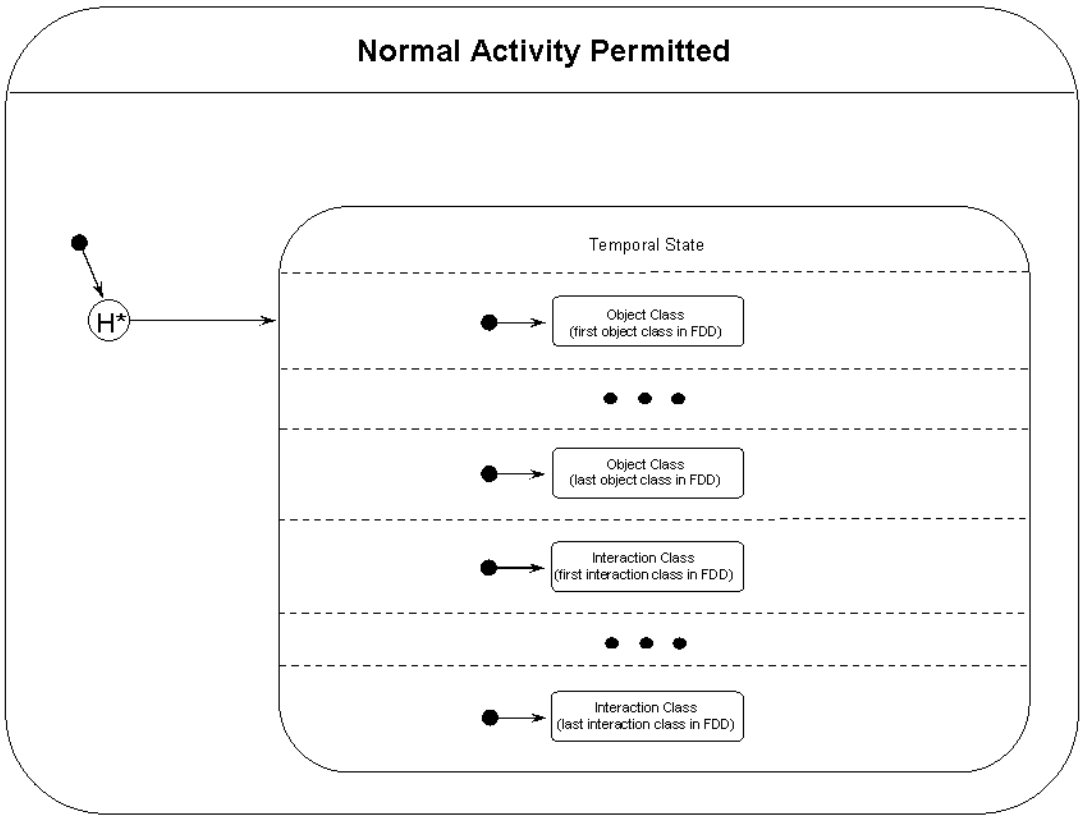

Figure 5—Normal Activity Permitted state

4.1.2 Federation execution save and restore

Any joined federate in the federation execution may initiate a save by invoking the *Request Federation Save* service. If there is no timestamp argument provided with the invocation of this service, the RTI shall instruct all of the joined federates in the federation execution (including the requesting joined federate) to save state by invoking the *Initiate Federate Save †* service at all of these joined federates as soon as possible. If there is a timestamp argument provided, the RTI will invoke the *Initiate Federate Save †* service at each of the time-constrained joined federates when their value of logical time advances to the value provided, and it will invoke the *Initiate Federate Save †* service at all nontime-constrained joined federates as soon as all of the time-constrained joined federates are eligible to be instructed to save.

When a joined federate receives an *Initiate Federate Save †* service invocation and subsequently saves its state, it shall use the federation save label (that was specified by the joined federate requesting the save in the *Request Federation Save* service), its federate type (that it specified when it joined the federation execution), and its federate designator (that was returned when it joined the federation execution) to distinguish the saved information.

Once a federation execution save has begun, an RTI shall save all queued messages that have not been delivered to the intended joined federates (e.g., future timestamp order (TSO) messages or receive order (RO) messages that were sent by other joined federates not yet in the Federate Save in Progress state). All saved information (by both the RTI and the joined federates) shall be persistent, i.e., it is stored onto disk or some other persistent medium, and it remains intact even after the federation execution is destroyed.

The saved information can be used at a later date, by some new set of joined federates, to restore all joined federates in the federation execution to the state in which they were when the save was accomplished. The federation can then resume effecting the execution from that saved point. Once a restore has begun, any messages that have not been delivered to the intended joined federates in the restoring federation execution shall be lost. After a restore is complete, the RTI shall deliver all related saved messages as appropriate to the joined federates in the restored federation execution (e.g., TSO messages at the appropriate logical time or RO messages as soon as possible in wall-clock time).

The set of federates joined to a federation execution when state is restored from a previously saved state need not be the exact set of federates that were joined to the federation execution when the state being restored was saved. The number of federates of each federate type that are joined to the federation execution, however, shall be the same. The federate-type argument supplied in the *Join Federation Execution* service invocation, therefore, is crucial to the save-and-restore process. Declaring a federate to be of a given type shall be equivalent to asserting that the joined federate can be restored using the state information saved by any other joined federate of that type.

There is no requirement that a save taken by one RTI implementation be restorable by another RTI implementation.

4.1.3 Synchronization discussion

Figure 6 depicts the state of a federation execution with respect to synchronization points. In order to fully explain synchronization points, both the state of the RTI and the state of each joined federate are detailed. Because multiple joined federates are involved, each transition within this set of state charts includes an argument denoting to which joined federate a service invocation applies (i.e., an argument of F_s denotes the *s*th joined federate).

The RTI shall keep track of each synchronization point separately, and the treatment of an arbitrary synchronization point by the RTI is expanded in Figure 7. Not shown in this diagram is how the RTI calculates the synchronization set (shown as *synch_set* within the state chart); this information is detailed in the service descriptions. Use of the synchronization set, however, is depicted in the state chart on the transition from the Synchronizing (s, L) states to the Synch Point Not Registered state. This transition denotes, by use of the universal quantifier over the synchronization set in the guard, that synchronization throughout a federation is achieved only when all joined federates in the synchronization set have concluded their individual synchronizations. A joined federate's individual synchronization, from the RTI's point of view, is detailed in Figure 8.

The final state chart in the set describing synchronization points is Figure 9. This state chart depicts how an arbitrary joined federate keeps track of an arbitrary synchronization point. This diagram includes how a joined federate registers a synchronization point as well as how a joined federate actually synchronizes.

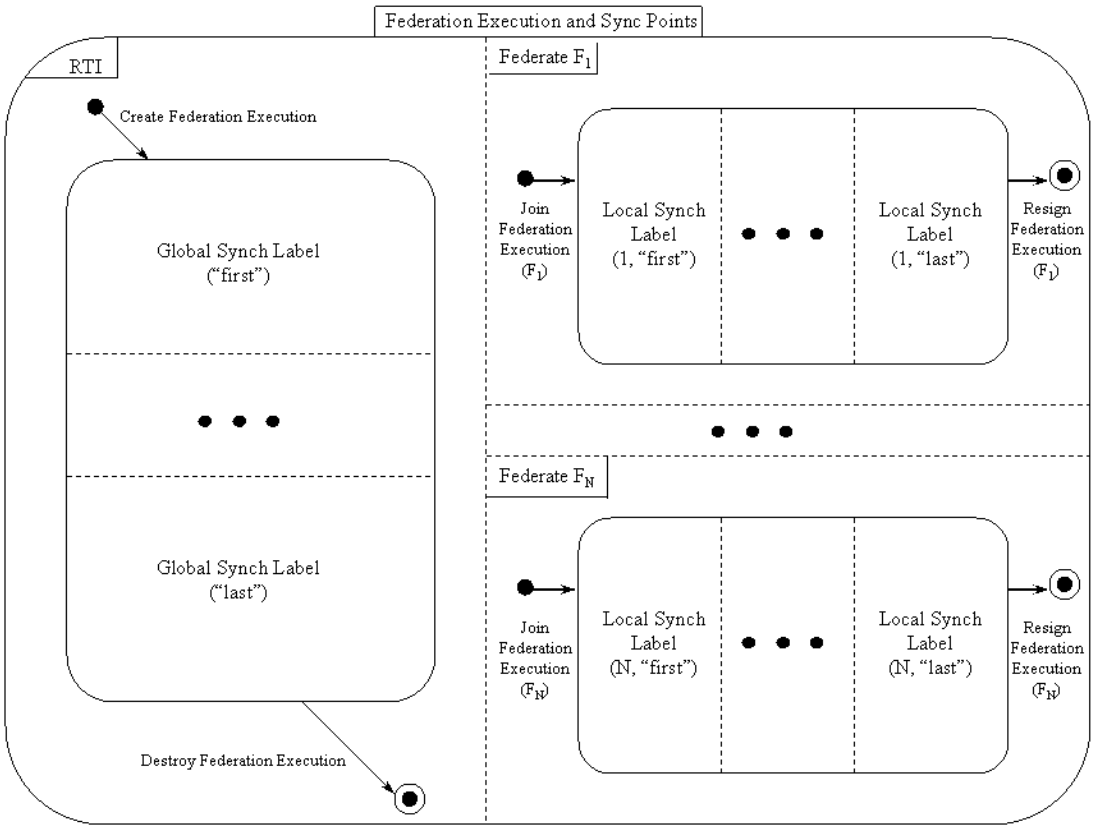

Figure 6—Federation execution and sync points

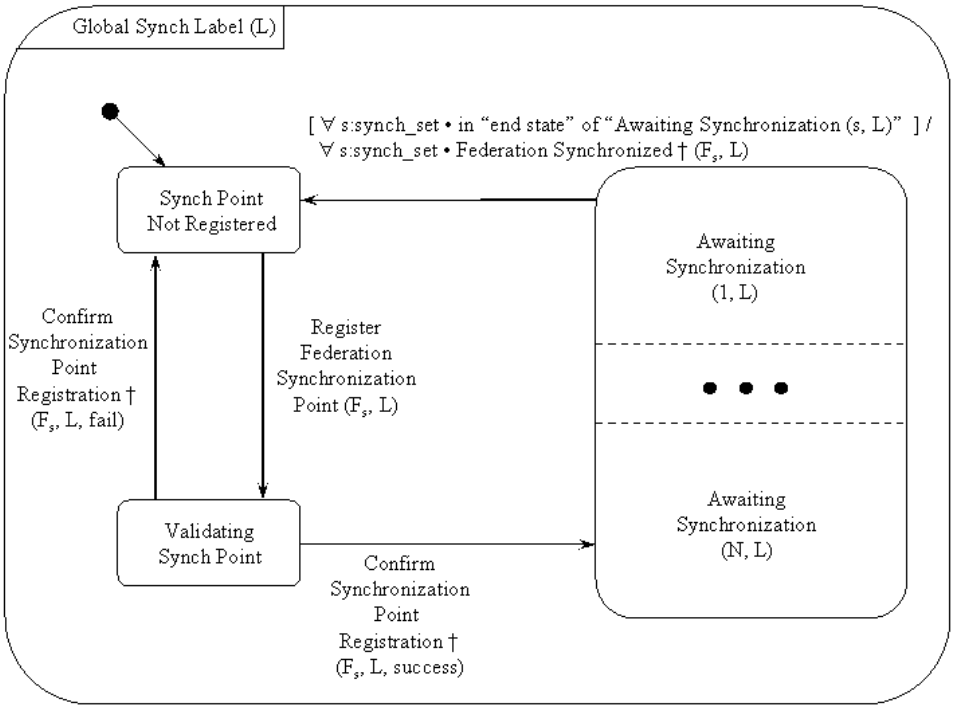

Figure 7—Global synch label (L)

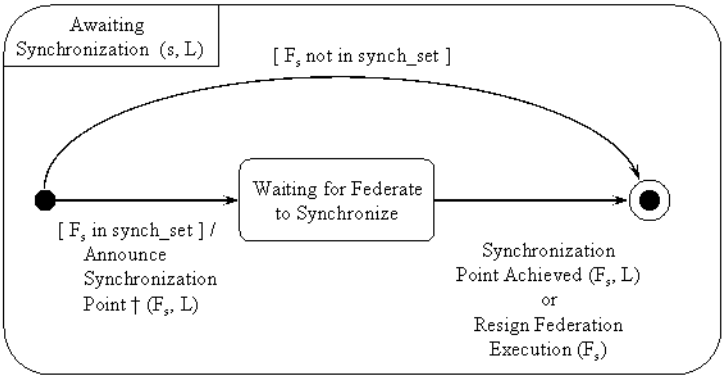

Figure 8—Awaiting synchronization (s, L)

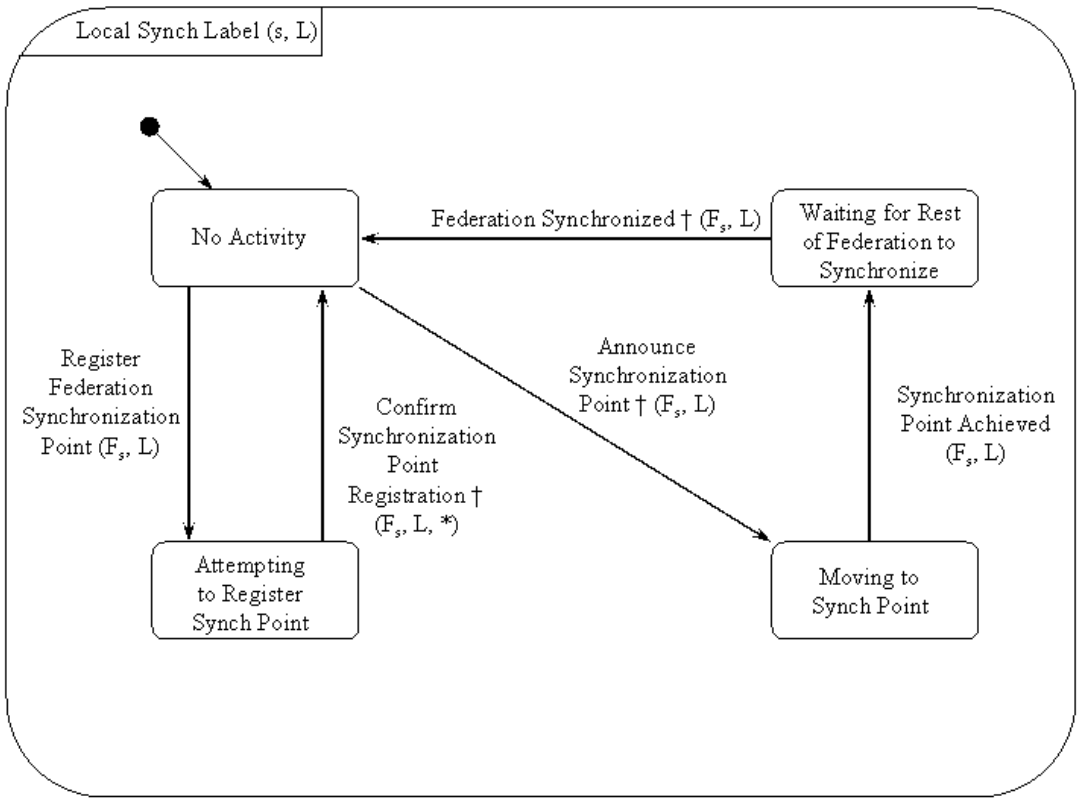

Figure 9—Local synch label (s, L)

4.1.4 FOM Document Data (FDD)

The FDD is data and information extracted and combined from FOM modules in order to support functionality described in this specification. The FOM module format and rules for combination are defined in IEEE Std 1516.2-2010. The FDD that the RTI retains and combines from the FOM modules shall be, at a minimum, the following:

- Object class structure table (complete)
- Interaction class structure table (complete)
- Attribute table (object, attribute, available dimensions, transportation, and order columns)
- Parameter table (interaction, parameter, available dimensions, transportation, and order columns)
- Dimension table (complete)
- Transportation type table (name and reliability columns)
- Update rate table (name and maximum update rate columns)
- Switches table (complete)

All the names within an FDD shall be case sensitive.

When the *Create Federation Execution* service is invoked, the RTI shall initialize the federation execution using FDD first from the MIM and then from the specified FOM modules. A user-extended MIM may be provided; otherwise, the RTI shall provide the standard MIM automatically.

Additional FOM modules may be provided in the *Join Federation Execution* service invocations. The FOM module designator arguments supplied to the *Create Federation Execution* service and the *Join Federation Execution* service shall designate compliant FOM modules (IEEE P1516/D5 and IEEE Std 1516.2-2010). *Create Federation Execution* and *Join Federation Execution* service invocations shall reject, issuing an exception, any entity that is not a compliant FOM module.

4.1.4.1 Rules for combining information from FOM modules

The RTI shall combine the FDD data of the MIM and FOM modules according to the rules for combining FOM data in IEEE Std 1516.2-2010 when the *Create Federation Execution* service is invoked as well as whenever the set of FOM modules has changed due to an invocation of the *Join Federation Execution* service. The RTI shall consider only the FDD when combining FOM modules. In case the FDD of the FOM modules provided in such a service call cannot be successfully combined or the invocation fails for any other reason, the FDD provided in the call shall not be included in the current FOM.

The RTI shall reject, issuing an exception, any attempt to load a FOM module for the following cases:

- When trying to define an object class, already existing in the current FOM, that is not a scaffolding class or that has a different set of class attributes compared to the current FOM. The set of class attributes is considered different from the current FOM if the sets are not identical in number of attributes or the attributes are not identical in name, available dimensions, transportation, or order.
- When trying to define an interaction class, already existing in the current FOM, that is not a scaffolding class or that has a different set of parameters compared to the current FOM. The set of parameters is considered different from the current FOM if the sets are not identical in number of parameters or the parameters are not identical in name, available dimensions, transportation, or order.
- When trying to define dimension, already existing in the current FOM, with a different definition compared to the current FOM.
- When trying to define transportation type, already existing in the current FOM, with a different reliability compared to the current FOM. (Comment note - description is not part of FDD nor should it be and thus cannot be used for rejection.)
- When trying to define update rate, already existing in the current FOM, with a different maximum rate compared to the current FOM.
- When trying to define switches, already existing in the current FOM, with a different definition.
- When the MIM or FOM module is noncompliant.

If a FOM module or MIM contains any definition that is rejected by the RTI, the RTI shall reject the FOM module or MIM in its entirety and shall not add any element to the FDD.

The union of all FOM modules in a given *Create Federation Execution* or *Join Federation Execution* service when added to the previously existing current FOM shall not result in any object class or interaction class not having a full definition. If this is the case, the entire service invocation shall fail. In case any of the FOM modules provided was rejected by the RTI, the entire service invocation shall also fail.

After a *Create Federation Execution* or *Join Federation Execution* service has completed successfully, all the provided FOM modules shall be available in the current FOM, and the corresponding MOM information shall be updated to reflect this.

Furthermore, any FDD provided as per above, once available in the federation execution, shall be available for the remaining life span of the federation execution.

4.1.4.2 Access to Current FOM information

The RTI shall provide access to the current FDD, the content of individual FOM modules, as well as the MIM. The following MOM interactions are provided:

a) For the federation execution:
   1) The attribute HLAFOMmoduleDesignatorList provides a list of designators for the FOM modules that specified the current FOM. Only FOM modules that resulted in the addition of one or more classes, dimensions, transportations, or update rates shall be included. Specifically, if several FOM modules with identical content have been specified, only the first shall be included and then only if it satisfies the preceding requirement.
   2) The attribute HLAfederation.HLAMIMDesignator provides the designator for the MIM used in the current FOM.
   3) The attribute HLAcurrentFDD provides the contents of the current FDD.
   4) The interactions HLArequestFOMmoduleData and HLAreportFOMmoduleData provide the contents of a specified FOM module.
   5) The interactions HLArequestMIMData and HLAreportMIMData provides the content of the MIM.
b) For each federate:
   1) The attribute HLAfederate.HLAFOMmoduleDesignatorList provides a list of designators for the FOM modules that were specified by that federate when invoking the *Create Federation Execution* and *Join Federation Execution* service.
   2) The interactions HLArequestFOMmoduleData and HLAreportFOMmoduleData provide the contents of a specified FOM module.

A FOM module designator shall be equal to the string (file name or URL) as supplied as part of the *Create Federation Execution* or *Join Federation Execution* service invocation. The FOM module designator for the HLAstandardMIM as supplied automatically by the RTI shall be "HLAstandardMIM."

4.1.5 Fault tolerance support

At any point during the execution, a fault may occur. A fault is defined as an event that prevents the entire federation from executing in an HLA-compliant manner. As a result, one or more federates may be disconnected from the federation. The RTI shall then send a HLAreportFederateLost MOM interaction. If the lost federate was time regulating, this interaction shall also provide the last known good logical timestamp to which the federate was granted.

The RTI shall then perform a resign on behalf of the lost federate, using the current Automatic Resign Directive, and then restart all federation-wide synchronized operations with the remaining federates. MOM data and advisories shall be updated to reflect the new state.

All TSO messages sent by the lost federate no later than the last known good logical timestamp before the fault shall be delivered. TSO messages sent by the lost federate after this point may or may not reach subscribing federates. RO messages from the lost federate may or may not reach subscribing federates.

The RTI shall attempt to notify any lost federate that it has been disconnected and is no longer a member of the federation using the *Connection Lost †* service invocation. The federate will then enter the Not Connected state (see Figure 3). Since the federate or its environment is subject to a fault, invoking the *Connection Lost †* service and entering the Not Connected state may or may not be attempted and may or may not be successful.

The disconnected federate may try to connect to the federation again using the regular connect and join sequence. This attempt may or may not be successful.

The fault detection methods and criteria depend on the technical implementation of the RTI and are not part of the HLA standard. A fault is an event that prevents the federation execution from continuing in an HLA-compliant manner. In particular, a fault may result in one or more federates being disconnected or lost from the federation. Such a federate shall be referred to as a lost federate. As a result of detection of a fault, the RTI shall send a separate HLAreportFederateLost MOM interaction identifying each lost federate. If a lost federate was time regulating, this interaction shall provide its last known good logical timestamp. The last known good logical timestamp shall be determined by the RTI for a lost federate so that all reliable TSO messages sent by the federate before the fault with timestamps less than or equal to the last known good logical timestamp are deliverable to federates that remain in the federation. After the HLAreportFederateLost interaction has been sent, the RTI shall process a resign on behalf of the lost federate, using the federate's current Automatic Resign Directive. The resignation of the federate shall result in the restart of all federation-wide synchronized operations with the remaining federates as if the lost federate itself had resigned. MOM data and advisories shall be updated to reflect the new state. TSO messages sent by the lost federate after the last known good timestamp may or may not be delivered to subscribing federates. RO messages from the lost federate may or may not be delivered to subscribing federates. When a federate becomes lost, the RTI may notify it using the *Connection Lost †* service invocation. The invocation of the *Connection Lost †* service by the RTI on the federate shall enter the federate into the Not Connected state (see Figure 3). Because a lost federate or its environment is subject to a fault, a federate that has been reported to the federation execution as lost but has not received the *Connection Lost †* service invocation is in an undefined state. A lost federate may try to connect to the RTI and join the federation execution again using the connect and join sequence. The attempts may or may not be successful, depending on the fault and steps taken outside of HLA to repair it. A fault may divide the federation into multiple independent segments, each having the potential to continue an HLA-compliant federation execution. Should this event occur, the RTI shall select exactly one such segment to continue the federation execution. Federates contained in all other segments shall be considered lost. It may be that the fault is catastrophic with no segment remaining to continue the federation execution. Each RTI implementation will define a mechanism for making this decision.

4.1.6 Logical time implementation

A federation execution must share a common implementation of logical time in order for joined federates and the RTI to have meaningful interactions involving logical time. No universal implementation of logical time has been proposed that will satisfy all federation executions. When a federation execution is created, a logical time implementation shall be identified. RTIs shall provide two default implementations of logical time, one integer and one floating point. The federate creating a federation execution may provide its own implementation of logical time for use in the federation execution.

4.2 *Connect* service

The *Connect* service shall be used to establish a connection between the unjoined federate and the RTI. No interaction between the unjoined federate and the RTI may take place until a successful invocation of the *Connect* service has been performed. If the optional local settings designator is not present, the default local settings shall be used.

The callback model argument indicates the desired callback model, either IMMEDIATE or EVOKED.

4.2.1 Supplied arguments

a) Callback model.
b) Optional local settings designator.

4.2.2 Returned arguments

a) None.

4.2.3 Preconditions

a) Not connected.

4.2.4 Postconditions

a) The unjoined federate is connected to the RTI.

4.2.5 Exceptions

a) Connection failed.
b) Unsupported callback model.
c) Invalid local settings designator.
d) Already connected.
e) RTI internal error.

4.3 *Disconnect* service

The *Disconnect* service shall be used to terminate the connection between a federate and the RTI.

The federate cannot be a member of the execution.

4.3.1 Supplied arguments

a) None.

4.3.2 Returned arguments

a) None.

4.3.3 Preconditions

a) The federate is connected to the RTI.
b) Federate is not a member of the execution.

4.3.4 Postconditions

a) The federate is disconnected from the RTI.

4.3.5 Exceptions

a) Federate is a member of the execution.
b) Not connected.
c) RTI internal error.

4.4 *Connection Lost †* service

The *Connection Lost †* service shall notify a previously joined federate that it has entered the Disconnected state as the result of a fault. Being disconnected, the federate is no longer joined to the federation execution.

The RTI will attempt the invocation of this service on a best-effort basis; however, it may fail since the RTI or federate may be in a faulty or invalid state.

4.4.1 Supplied arguments

a) Text description of the fault.

4.4.2 Returned arguments

a) None.

4.4.3 Preconditions

a) The joined federate is connected.

4.4.4 Postconditions

a) Federate is notified that it has entered the Disconnected state.

4.4.5 Exceptions

a) Federate internal error.

4.5 *Create Federation Execution* service

The *Create Federation Execution* service shall create a new federation execution and add it to the set of supported federation executions. Each federation execution created by this service shall be independent of all other federation executions, and there shall be no intercommunication within the RTI between federation executions.

The RTI shall load the supplied MIM if specified; otherwise, it shall automatically load the standard MIM (see Annex G). The MIM shall be loaded before any supplied FOM module is loaded. If a MIM argument is supplied, the supplied MIM module designator shall not be "HLAstandardMIM."

At least one FOM module shall be supplied.

The FOM modules supplied, together with the default or supplied MIM, shall result in a compliant FDD. The FOM module designators argument identifies the FOM modules that, together with the MIM, shall furnish the FDD for the federation execution to be created.

4.5.1 Supplied arguments

a) Federation execution name.
b) Set of FOM module designators.
c) Optional MIM designator.
d) Optional logical time implementation. If not provided, the RTI provided HLAfloat64Time representation of logical time shall be used.

4.5.2 Returned arguments

a) None.

4.5.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution does not exist.

4.5.4 Postconditions

a) A federation execution, with the given name, exists and may be joined by federates.

4.5.5 Exceptions

a) Could not create logical time implementation
b) Invalid FOM module.
c) Could not locate FOM module indicated by supplied designator.
d) MIM designator shall not be HLAstandardMIM.
e) Invalid MIM.
f) Could not locate MIM indicated by supplied designator.
g) The specified federation execution already exists.
h) Not connected.
i) RTI internal error.

4.5.6 Reference state charts

a) Figure 1.
b) Figure 6.

4.6 *Destroy Federation Execution* service

The *Destroy Federation Execution* service shall remove a federation execution from the RTI set of supported federation executions. All federation activity shall have stopped, and there shall be no joined federates (all joined federates shall have resigned, either by explicit action or via MOM activity) before this service is invoked.

4.6.1 Supplied arguments

a) Federation execution name.

4.6.2 Returned arguments

a) None.

4.6.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) There are no federates joined to this federation execution.

4.6.4 Postconditions

a) The federation execution does not exist.

4.6.5 Exceptions

a) Federates are joined to the federation execution.
b) The federation execution does not exist.
c) Not connected.
d) RTI internal error.

4.6.6 Reference state charts

a) Figure 1.
b) Figure 6.

4.7 *List Federation Executions* service

The *List Federation Executions* service shall be used to request a list of the currently existing federation executions.

4.7.1 Supplied arguments

a) None.

4.7.2 Returned arguments

a) None.

4.7.3 Preconditions

a) The federate is connected to the RTI.

4.7.4 Postconditions

a) None.

4.7.5 Exceptions

a) Not connected.
b) RTI internal error.

4.8 *Report Federation Executions †* service

The *Report Federation Executions †* service provides a list of the currently existing federation executions.

4.8.1 Supplied arguments

a) List of pairs of federation executions and time representations.

4.8.2 Returned arguments

a) None.

4.8.3 Preconditions

a) None.

4.8.4 Postconditions

a) None.

4.8.5 Exceptions

a) Federate internal error.

4.9 *Join Federation Execution* service

The *Join Federation Execution* service shall affiliate the federate with a federation execution. Invocation of the *Join Federation Execution* service shall indicate the intention to participate in the specified federation. The federate name argument shall be unique within a federation execution at any given time. It may, however, be assigned to a different federate after a restore operation. The RTI shall provide a unique name if no federate name is provided, and it is the responsibility of the federate to obtain and preserve this unique name for potential restore operations. The federate type argument shall distinguish federate categories for federation save-and-restore purposes. 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.

In addition, the *Join Federation Execution* service may also be used to provide additional FOM module designators. The optional set of additional FOM module designators argument identifies the FOM modules that provide additional FDD. Contents of the FOM modules may duplicate information in the current FDD of the federation execution, but they shall not conflict with the current FDD.

4.9.1 Supplied arguments

a) Optional federate name. If not provided, the RTI shall assign a unique name.
b) Federate type.
c) Federation execution name.
d) Optional set of additional FOM module designators.

4.9.2 Returned arguments

a) Joined federate designator.

4.9.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is not joined to that federation execution.
d) Federate name not already in use.
e) Federate save not in progress.
f) Federate restore not in progress.

4.9.4 Postconditions

a) The joined federate is a member of the federation execution.

4.9.5 Exceptions

a) Could not create logical time implementation.
b) Federate name already in use.
c) The specified federation execution does not exist.

d) Invalid FOM module.
e) Inconsistent FOM module.
f) Could not locate FOM module indicated by supplied designator.
g) Federate save in progress.
h) Federate restore in progress.
i) The federate is already joined to the federation execution.
j) Not connected.
k) RTI internal error.

4.9.6 Reference state charts

a) Figure 1.
b) Figure 3.
c) Figure 6.

4.10 *Resign Federation Execution* service

The *Resign Federation Execution* service shall indicate the requested cessation of federation participation. Before resigning, ownership of instance attributes held by the joined federate should be resolved. The joined federate may transfer ownership of these instance attributes to other joined federates, unconditionally divest them for ownership acquisition at a later point, or delete the object instance of which they are a part (assuming the joined federate has the privilege to delete these object instances). As a convenience to the joined federate, the *Resign Federation Execution* service shall accept an action argument that directs the RTI to perform zero or more of the following actions:

— Unconditionally divest all owned instance attributes for future ownership acquisition. This action shall place the instance attributes into an unowned state (implying that their values are not being updated), which shall make them eligible for ownership by another joined federate. See Clause 7 for a more detailed description.
— Delete all object instances for which the joined federate has that privilege (implied invocation of the *Delete Object Instance* service).
— Cancel all pending instance attribute ownership acquisitions. The use of this directive may interfere with the intended semantics of negotiated instance attribute ownership divestiture by allowing instance attributes divested in this way to be unowned (because the cancellation directive may not succeed).

If a federate invokes this service with either directive 1, 4, or 5 as listed in 4.10.1, then for each instance attribute that becomes unowned as a result, if no joined federates are in either the Acquiring state or the 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 state or the 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 joined federates that are both eligible to own the instance attribute and not in either the Acquiring state or the Willing to Acquire state with respect to the specified instance attribute. As long as the instance attribute remains unowned, the RTI shall try to identify joined federates that are willing to own the instance attribute; but once the instance attribute becomes owned, the RTI shall not invoke the *Request Attribute Ownership Assumption †* service at any additional federates. If no other federates are joined to the federation execution, then the resigning federate is the last joined federate, and RTI shall process directive 2 as listed in 4.10.1 regardless of supplied arguments.

4.10.1 Supplied arguments

a) Directive to
   1) Unconditionally divest ownership of all owned instance attributes.
   2) Delete all object instances for which the joined federate has the delete privilege.
   3) Cancel all pending instance attribute ownership acquisitions.
   4) Perform action 2) and then action 1).
   5) Perform action 3), action 2), and then action 1).
   6) Perform no actions.

4.10.2 Returned arguments

a) None.

4.10.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) If directive 2 (as listed in 4.10.1) is supplied, the joined federate does not own any instance attributes of object instances for which it does not also have the delete privilege.
e) If directive 3 or 6 (as listed in 4.10.1) is supplied, the joined federate does not own any instance attributes in the federation execution.
f) If directive 1, 2, 4, or 6 (as listed in 4.10.1) is specified, there are no instance attributes for which the joined federate has performed one of the following actions:
   1) Invoked the *Attribute Ownership Acquisition* service and has not yet received a corresponding invocation of the *Confirm Attribute Ownership Acquisition Cancellation †* service, the *Attribute Ownership Unavailable †* service, or the *Attribute Ownership Acquisition Notification †* service.
   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.
   3) Invoked the *Attribute Ownership Acquisition If Available* service and has subsequently invoked the *Attribute Ownership Acquisition* service (after which action 1 applies)

4.10.4 Postconditions

a) The federate is not a member of the federation execution.
b) There are no instance attributes in the federation execution owned by the joined federate.
c) If directive 2, 4, or 5 (as listed in 4.10.1) is supplied, all object instances for which the joined federate has the delete privilege are deleted.
d) If directive 3 or 5 (as listed in 4.10.1) is supplied, all pending instance attribute ownership acquisitions for the joined federate are canceled.

4.10.5 Exceptions

a) Invalid resign directive.
b) Cannot resign due to pending attempt to acquire instance attribute ownership.
c) The joined federate owns instance attributes.
d) The federate is not a federation execution member.
e) The specified federation execution does not exist.
f) Not connected.
g) RTI internal error.

4.10.6 Reference state charts

a) Figure 1.
b) Figure 3.
c) Figure 6.
d) Figure 8.

4.11 *Register Federation Synchronization Point* service

The *Register Federation Synchronization Point* service shall be used to initiate the registration of an upcoming synchronization point label. When a synchronization point label has been successfully registered (indicated through the *Confirm Synchronization Point Registration †* service), the RTI shall inform some or all joined federates of the label's existence by invoking the *Announce Synchronization Point †* service at those joined federates. The optional set of joined federate designators shall be provided by the joined federate if only a subset of joined federates in the federation execution are to be informed of the label existence. If the set is provided, joined federates in the federation execution, which are members of the set, shall be informed of the label existence. The set may contain valid federate designators of federates no longer joined in the federation execution. If the optional set of joined federate designators is empty or not supplied, all joined federates in the federation execution shall be informed of the label's existence. If the optional set of designators is not empty, all designated joined federates shall be federation execution members. The user-supplied tag shall provide a vehicle for information to be associated with the synchronization point and shall be announced along with the synchronization label. It is possible for multiple synchronization points registered by the same or different joined federates to be pending at the same point.

4.11.1 Supplied arguments

a) Synchronization point label.
b) User-supplied tag.
c) Optional set of joined federate designators.

4.11.2 Returned arguments

a) None.

4.11.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) If an optional set of joined federate designators is supplied, those federate designators shall be valid.
e) The supplied synchronization point label is not in use.
f) Federate save not in progress.
g) Federate restore not in progress.

4.11.4 Postconditions

a) The synchronization label is known to the RTI.

4.11.5 Exceptions

a) Federate save in progress.
b) Federate restore in progress.

c) The federate is not a federation execution member.
d) Not connected.
e) RTI internal error.

4.11.6 Reference state charts

a) Figure 7.
b) Figure 9.

4.12 *Confirm Synchronization Point Registration †* service

The *Confirm Synchronization Point Registration †* service shall indicate to the requesting joined federate the status of a requested federation synchronization point registration. This service shall be invoked in response to a *Register Federation Synchronization Point* service invocation. A registration-success indicator argument indicating success shall mean the label has been successfully registered.

If the registration-success indicator argument indicates failure, the failure reason argument shall be provided to identify the reason that the synchronization point registration failed. Possible reasons for the synchronization point registration failure are the following:

— The specified label is already in use.
— A synchronization set member is not a joined federate.

A registration attempt that ends with a negative success indicator shall have no other effect on the federation execution.

4.12.1 Supplied arguments

a) Synchronization point label.
b) Registration-success indicator.
c) Optional failure reason.

4.12.2 Returned arguments

a) None.

4.12.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate has invoked *Register Federation Synchronization Point* service for the specified label.
d) Federate save not in progress.
e) Federate restore not in progress.

4.12.4 Postconditions

a) If the registration-success indicator is positive, the specified label and associated user-supplied tag shall be announced to the appropriate joined federates.
b) If the registration-success indicator is negative, this service and the corresponding *Register Federation Synchronization Point* service invocation have no consequence.

4.12.5 Exceptions

a) Federate internal error.

4.12.6 Reference state charts

a) Figure 7.
b) Figure 9.

4.13 *Announce Synchronization Point †* service

The *Announce Synchronization Point †* service shall inform a joined federate of the existence of a new synchronization point label. When a synchronization point label has been registered with the *Register Federation Synchronization Point* service, the RTI shall invoke the *Announce Synchronization Point †* service, either at all the joined federates in the federation execution or at the specified set of joined federates, to inform them of the label existence. The joined federates informed of the existence of a given synchronization point label via the *Announce Synchronization Point †* service shall form the synchronization set for that point. If the optional set of joined federate designators was null or not provided when the synchronization point label was registered, the RTI shall also invoke the *Announce Synchronization Point †* service at all federates that join the federation execution after the synchronization label was registered but before the RTI has ascertained that all joined federates that were informed of the synchronization label existence have invoked the *Synchronization Point Achieved* service. These newly joining federates shall also become part of the synchronization set for that point. Joined federates that resign from the federation execution after the announcement of a synchronization point but before the federation synchronizes at that point shall be removed from the synchronization set. The user-supplied tag supplied by the *Announce Synchronization Point †* service shall be the tag that was supplied to the corresponding *Register Federation Synchronization Point* service invocation.

4.13.1 Supplied arguments

a) Synchronization point label.
b) User-supplied tag.

4.13.2 Returned arguments

a) None.

4.13.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The synchronization point has been registered.
d) Federate save not in progress.
e) Federate restore not in progress.

4.13.4 Postconditions

a) The synchronization label is known to the joined federate and may be used in the *Synchronization Point Achieved* and *Federation Synchronized †* services.

4.13.5 Exceptions

a) Federate internal error.

4.13.6 Reference state charts

a) Figure 8.
b) Figure 9.

4.14 *Synchronization Point Achieved* service

The *Synchronization Point Achieved* service shall inform the RTI that the joined federate has reached the specified synchronization point. Once all joined federates in the synchronization set for a given point have invoked this service, the RTI shall not invoke the *Announce Synchronization Point †* on any newly joining federates. The optional synchronization-success indicator shall inform the RTI that the joined federate was synchronized either successfully or unsuccessfully. The RTI shall consider the absence of the indicator as if the joined federate synchronized successfully.

4.14.1 Supplied arguments

a) Synchronization point label.
b) Optional synchronization-success indicator.

4.14.2 Returned arguments

a) None.

4.14.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The synchronization point has been announced at this federate.
e) Federate save not in progress.
f) Federate restore not in progress.

4.14.4 Postconditions

a) The joined federate is noted as having reached the specified synchronization point.

4.14.5 Exceptions

a) The synchronization point label is not announced.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

4.14.6 Reference state charts

a) Figure 8.
b) Figure 9.

4.15 *Federation Synchronized †* service

The *Federation Synchronized †* service shall inform the joined federate that all joined federates in the synchronization set of the specified synchronization point have invoked the *Synchronization Point Achieved* service for that point. This service shall be invoked at all joined federates that are in the synchronization set for that point to indicate that the joined federates in the synchronization set have synchronized at that point. Once the synchronization set for a point synchronizes (i.e., the *Federation Synchronized †* service has been invoked at all joined federates in the set), that point shall no longer be registered, and the synchronization set for that point shall no longer exist. The set of joined federate designators indicates which federates in the synchronization set invoked the *Synchronization Point Achieved* service with an indication of an unsuccessful synchronization (via the synchronization-success indicator).

4.15.1 Supplied arguments

a) Synchronization point label.
b) Set of joined federate designators.

4.15.2 Returned arguments

a) None.

4.15.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The synchronization point has been registered.
d) The synchronization point has been announced at this federate.
e) All joined federates in the synchronization set have invoked the *Synchronization Point Achieved* service using the specified label.
f) Federate save not in progress.
g) Federate restore not in progress.

4.15.4 Postconditions

a) The joined federate is informed that all joined federates in the synchronization set, including it if appropriate, have invoked the *Synchronization Point Achieved* service using the specified label.

4.15.5 Exceptions

a) Federate internal error.

4.15.6 Reference state charts

a) Figure 7.
b) Figure 9.

4.16 *Request Federation Save* service

The *Request Federation Save* service shall specify that a federation save should take place.

If the optional timestamp argument is not present, the RTI shall instruct all federation execution members to save state as soon as possible after the invocation of the *Request Federation Save* service.

An optional timestamp argument may be supplied only if the joined federate is time-regulating or if the Greatest Available Logical Time (GALT) is defined for the joined federate. Supplied timestamp arguments shall be larger than the logical time of all time-constrained joined federates in the federation execution. For time-regulating joined federates, this is ensured by using only timestamps that are valid for the joined federate to send in a TSO message (see 8.1 for more information on valid TSO message timestamps). For nonregulating joined federates at which GALT is defined, this can be ensured by only using timestamps that are greater than the joined federate's GALT value.

If a valid optional timestamp argument is present, the RTI shall instruct each time-constrained joined federate to save state (via the *Initiate Federate Save †* service) as soon as the joined federate has received all messages that it will receive as TSO messages that have timestamps less than or equal to the timestamp of the scheduled save[6] and either

— The joined federate is in the Time Advancing state as a result of invocation of a *Time Advance Request Available*, *Next Message Request Available*, or *Flush Queue Request* service and the logical time of the next grant is greater than the timestamp of the scheduled save, or
— The joined federate is in the Time Advancing state as a result of invocation of either a *Time Advance Request* or *Next Message Request* service and the logical time of the next grant is greater than or equal to the timestamp of the scheduled save

When all time-constrained joined federates are eligible to be instructed to save, the RTI shall instruct all nontime-constrained joined federates to save state.

The RTI shall instruct a joined federate to save state by invoking the *Initiate Federate Save †* service at that joined federate.

At most, one requested save shall be outstanding. A new save request shall replace any outstanding save request. However, a save request cannot happen during a federate save in progress, which is between the RTI invocation of the *Initiate Federate Save †* service and the RTI invocation of the *Federation Saved †* service.

4.16.1 Supplied arguments

a) Federation save label.
b) Optional timestamp.

4.16.2 Returned arguments

a) None.

4.16.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) If an optional timestamp is supplied, either
    1) The joined federate is time-regulating, and the timestamp is a valid timestamp for the joined federate to send in a TSO message, or
    2) GALT is defined for the joined federate, and the timestamp is greater than the joined federate's GALT value.
e) Federate save not in progress.
f) Federate restore not in progress.

[6]In other words, the federate's GALT will always be greater than the timestamp of the scheduled save.

4.16.4 Postconditions

a) A federation save has been requested.
b) All previous requested saves are canceled.

4.16.5 Exceptions

a) Timestamp has already passed (if optional timestamp argument supplied).
b) Timestamp is invalid (if optional timestamp argument is supplied).
c) Federate may not use the timestamp argument (if optional timestamp argument is supplied).
d) Federate save in progress.
e) Federate restore in progress.
f) The federate is not a federation execution member.
g) Not connected.
h) RTI internal error.

4.16.6 Reference state charts

a) Figure 3.

4.17 *Initiate Federate Save †* service

The *Initiate Federate Save †* service shall instruct the joined federate to save state. The joined federate should save as soon as possible after the invocation of the *Initiate Federate Save †* service. The label provided to the RTI when the save was requested, via the *Request Federation Save* service, shall be supplied to the joined federate. The joined federate shall use this label, the name of the federation execution, its joined federate designator, and its federate type, which it supplied when it invoked the *Join Federation Execution* service, to distinguish the saved state information. If a joined federate is not time constrained, it shall expect to receive an *Initiate Federate Save †* service invocation at any point. If a joined federate is time-constrained, it shall expect to receive an *Initiate Federate Save †* service invocation only when it is in the Time Advancing state. The joined federate shall stop providing new information to the federation immediately after receiving the *Initiate Federate Save †* service invocation. The joined federate may resume providing new information to the federation only after receiving the *Federation Saved †* service invocation.

If the corresponding *Request Federation Save* service invocation had a timestamp argument, that timestamp shall be provided to the resulting *Initiate Federate Save†* service invocation.

4.17.1 Supplied arguments

a) Federation save label.
a) Optional timestamp.

4.17.2 Returned arguments

a) None.

4.17.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) A federation save has been scheduled via the *Request Federation Save* service.
d) Federate save not in progress.
e) Federate restore not in progress.

4.17.4 Postconditions

a) The joined federate has been instructed to begin saving its state.

4.17.5 Exceptions

a) Federate internal error.

4.17.6 Reference state charts

a) Figure 3.

4.18 *Federate Save Begun* service

The *Federate Save Begun* service shall notify the RTI that the joined federate is beginning to save its state.

4.18.1 Supplied arguments

a) None.

4.18.2 Returned arguments

a) None.

4.18.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The joined federate has received an *Initiate Federate Save †* invocation.
e) The joined federate is ready to start saving its state.
f) Federate restore not in progress.

4.18.4 Postconditions

a) The RTI has been informed that the joined federate has begun saving its state.

4.18.5 Exceptions

a) Save not initiated.
b) Federate restore in progress.
c) The federate is not a federation execution member.
d) Not connected.
e) RTI internal error.

4.18.6 Reference state charts

a) Figure 3.

4.19 *Federate Save Complete* service

The *Federate Save Complete* service shall notify the RTI that the joined federate has completed its save attempt. The save-success indicator shall inform the RTI that the joined federate save either succeeded or failed.

4.19.1 Supplied arguments

a) Federate save-success indicator.

4.19.2 Returned arguments

a) None.

4.19.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The joined federate has invoked the *Federate Save Begun* service for this save.
e) The joined federate has completed the attempt to save its state.
f) Federate restore not in progress.

4.19.4 Postconditions

a) The RTI has been informed of the status of the state save attempt.

4.19.5 Exceptions

a) Federate has not begun save.
b) Federate restore in progress.
c) The federate is not a federation execution member.
d) Not connected.
e) RTI internal error.

4.19.6 Reference state charts

a) Figure 3.

4.20 *Federation Saved †* service

The *Federation Saved †* service shall inform the joined federate that the federation save process is complete, and it shall indicate whether it completed successfully. A save-success indicator argument indicating success shall mean that all joined federates at which the *Initiate Federate Save †* service was invoked have invoked the *Federate Save Complete* service with a save-success indicator that indicated success.

If the save-success indicator argument indicates failure, the optional failure reason argument shall be provided to identify the reason that the federation save failed. Possible reasons for the failure of a federation save are as follows:

— RTI was unable to save.
— One or more joined federates at which the *Initiate Federate Save †* service was invoked have invoked the *Federate Save Complete* service with a save-success indicator that indicated failure.

— One or more joined federates at which the *Initiate Federate Save †* service was invoked have resigned from the federation execution.
— The RTI has detected failure at one or more of the joined federates at which the *Initiate Federate Save †* service was invoked.
— The optional timestamp supplied with the save request cannot be honored (due to possible race conditions in the distributed calculation of GALT).
— Federation save aborted.

All joined federates that received an invocation of the *Initiate Federate Save †* service shall receive an invocation of the *Federation Saved †* service. Also, a joined federate that has invoked the *Request Federation Save* service may receive an invocation of the *Federation Saved †* service without having first received an invocation of the *Initiate Federate Save †* service.

4.20.1 Supplied arguments

a) Federation save-success indicator.
b) Optional failure reason.

4.20.2 Returned arguments

a) None.

4.20.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) Federate restore not in progress.

4.20.4 Postconditions

a) The joined federate has been informed of the success or failure of the federation save attempt.
b) The joined federate may resume providing new information to the federation.

4.20.5 Exceptions

a) Federate internal error.

4.20.6 Reference state charts

a) Figure 3.

4.21 *Abort Federation Save* service

The *Abort Federation Save* service requests the RTI to abort the federation save that is underway.

The service takes no arguments and affects the current save. This service may be invoked only while the federate is in the Federate Save in Progress state. Upon return of the service invocation, a request to abort the current save has successfully been made.

The RTI shall abort the current save by invoking the *Federation Saved †* service on all federates that are in the Federate Save in Progress state, regardless of each individual federate's save substate. Joined federates not in the Federation Save in Progress state may not receive the *Initiate Federation Saved †* callback for this federation save. The save-success indicator for the *Federation Saved †* service shall indicate failure

(federate save aborted) if not all federates have successfully invoked the *Federate Save Complete* service with a save-success indicator that indicated success. Since a federate remains in the Federate Save in Progress state after it successfully invokes the *Federate Save Complete* service and before the *Federation Saved †* service is invoked on it by the RTI, it is possible that all other federates may have invoked the *Federate Save Complete* service with a save-success indicator of indicating success. In this case, it is possible that the *Federation Saved †* service is invoked at each federate with a save-success indicator indicating success after a invocation of the *Abort Federation Save* service.

4.21.1 Supplied arguments

a) None.

4.21.2 Returned arguments

a) None.

4.21.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The federate is in the Save in Progress state.

4.21.4 Postconditions

a) An abort federation save has been requested.

4.21.5 Exceptions

a) Federate save not in progress.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

4.21.6 Reference state charts

a) Figure 3.

4.22 *Query Federation Save Status* service

The *Query Federation Save Status* service shall be used to request the status of a federation save. The RTI shall provide the federation save status via a *Federation Save Status Response †* service invocation. No save label argument is needed since at most one federation save can be in progress.

4.22.1 Supplied arguments

a) None.

4.22.2 Returned arguments

a) None.

4.22.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) Federate restore not in progress.

4.22.4 Postconditions

a) The request for a federation save status has been received by the RTI.

4.22.5 Exceptions

a) Federate restore in progress.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

4.22.6 Reference state charts

a) None.

4.23 *Federation Save Status Response †* service

The *Federation Save Status Response †* service shall be used to provide the status of a federation save. This service shall be invoked by the RTI at a joined federate in response to a *Query Federation Save Status* service invocation by the same joined federate. This service shall supply a list of the joined federates participating in the federation save and the save status for each one. A joined federate save status shall be one of the following values:

— No federate save in progress
— Federate instructed to save
— Federate saving
— Federate waiting for federation to save

4.23.1 Supplied arguments

a) List of joined federates and save status for each.

4.23.2 Returned arguments

a) None.

4.23.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) Federate restore not in progress.
d) Joined federate requested save status via *Query Federation Save Status* service.

4.23.4 Postconditions

a) The joined federate has been informed of the status of the federation save.

4.23.5 Exceptions

a) Federate internal error.

4.23.6 Reference state charts

a) None.

4.24 *Request Federation Restore* service

The *Request Federation Restore* service shall direct the RTI to begin the federation execution restoration process. Federation restoration shall begin as soon after the validation of the *Request Federation Restore* service invocation as possible. A valid federation restoration request shall be indicated with the *Confirm Federation Restoration Request †* service.

4.24.1 Supplied arguments

a) Federation save label.

4.24.2 Returned arguments

a) None.

4.24.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The federation has a save with the specified label.
e) The correct number of joined federates of the correct types that were joined to the federation execution when the save was accomplished are a subset of the federates currently joined to the federation execution.
f) All previous *Request Federation Restore* service invocations from the joined federate have been acknowledged with a corresponding *Confirm Federation Restoration Request †* service.
g) Federate save not in progress.
h) Federate restore not in progress.

4.24.4 Postconditions

a) The RTI has been notified of the request to restore a former federation execution state.

4.24.5 Exceptions

a) Federate save in progress.
b) Federate restore in progress.
c) The federate is not a federation execution member.
d) Not connected.
e) RTI internal error.

4.24.6 Reference state charts

a) Figure 3.

4.25 *Confirm Federation Restoration Request †* service

The *Confirm Federation Restoration Request †* service shall indicate to the requesting joined federate the status of a requested federation restoration. This service shall be invoked in response to a *Request Federation Restore* service invocation. A positive request-success indicator informs the joined federate that RTI restoration state information has been located that corresponds to the following:

— The indicated label.
— Federation execution name and FDD of this federation execution.
— The census of joined federates matches in number and type the census of joined federates present when the save was taken.

Additionally, no other joined federate is currently attempting to restore the federation. Should more than one joined federate attempt to restore the federation at a given point, at most, one joined federate shall receive a positive indication through this service, and all others shall receive a negative indication. A federation restoration attempt that ends with a negative request-success indicator shall have no other effect on the federation execution.

4.25.1 Supplied arguments

a) Federation save label.
b) Request-success indicator.

4.25.2 Returned arguments

a) None.

4.25.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate has requested a federation restore via the *Request Federation Restore* service.
d) Federate restore not in progress.
e) Federate save not in progress.

4.25.4 Postconditions

a) If the request-success indicator is positive, federate restore in progress.
b) If the request-success indicator is positive, the federation has a saved state with the specified label.
c) If the request-success indicator is positive, the correct number of joined federates of the correct types that were joined to the federation execution when the save was accomplished are currently joined to the federation execution.
d) If the request-success indicator is negative, this service and the corresponding *Request Federation Restore* service invocation have no consequence.

4.25.5 Exceptions

a) Federate internal error.

4.25.6 Reference state charts

a) Figure 3.

4.26 *Federation Restore Begun †* service

The *Federation Restore Begun †* service shall inform the joined federate that a federation restoration is imminent. The joined federate shall stop providing new information to the federation immediately after receiving the *Federation Restore Begun †* service invocation. The joined federate may resume providing new information to the federation only after receiving the *Federation Restored †* service invocation. The RTI shall invoke this service at all joined federates, including the joined federate that requested the restore.

4.26.1 Supplied arguments

a) None.

4.26.2 Returned arguments

a) None.

4.26.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) Federate save not in progress.
d) Federate restore not in progress.

4.26.4 Postconditions

a) The joined federate has been instructed to stop providing new information to the federation.

4.26.5 Exceptions

a) Federate internal error.

4.26.6 Reference state charts

a) Figure 3.

4.27 *Initiate Federate Restore †* service

The *Initiate Federate Restore †* service shall instruct the joined federate to return to a previously saved state. The joined federate shall select the appropriate restoration state information based on the name of the current federation execution, the supplied federation save label, the supplied joined federate designator, and the supplied federate name. As a result of this service invocation, a joined federate's designator and name could change from the value supplied by the *Join Federation Execution* service.

4.27.1 Supplied arguments

a) Federation save label.
b) Joined federate designator.
c) Federate name.

4.27.2 Returned arguments

a) None.

4.27.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate has a save with the specified label.
d) Federate save not in progress.

4.27.4 Postconditions

a) The joined federate has been informed to begin restoring state.
b) The joined federate's designator and name may be changed.

4.27.5 Exceptions

a) Federate internal error.

4.27.6 Reference state charts

a) Figure 3.

4.28 *Federate Restore Complete* service

The *Federate Restore Complete* service shall notify the RTI that the joined federate has completed its restore attempt. If the restore was successful, the joined federate shall be in the state that either it or some other joined federate of its type was in when the federation save associated with the label occurred, with the distinction that the joined federate shall now be waiting for an invocation of the *Federation Restored †* service.

4.28.1 Supplied arguments

a) Federate restore-success indicator.

4.28.2 Returned arguments

a) None.

4.28.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The joined federate was directed to restore through invocation of the *Initiate Restore* service.
e) If the restore was successful, the joined federate is in a state identical to the state that either it or some other joined federate of its type was in when the federation save associated with the supplied label occurred, with the distinction that the joined federate is now waiting for an invocation of the *Federation Restored †* service.
f) If the restore was unsuccessful, the joined federate is in an undefined state.
g) Federate save not in progress.

4.28.4 Postconditions

a) The RTI has been informed of the status of the restore attempt.

4.28.5 Exceptions

a) Restore not requested.
b) Federate save in progress.
c) The federate is not a federation execution member.
d) Not connected.
e) RTI internal error.

4.28.6 Reference state charts

a) Figure 3.

4.29 *Federation Restored †* service

The *Federation Restored †* service shall inform the joined federate that the federation restore process is complete, and it shall indicate whether it completed successfully. A restore-success indicator argument indicating success shall mean that all joined federates at which the *Federation Restore Begun †* service was invoked have invoked the *Federate Restore Complete* service with a restore-success indicator that indicated success.

If the restore-success indicator argument indicates failure, the optional failure reason argument shall be provided to identify the reason that the federation restore failed. Possible reasons for the failure of a federation restore are as follows:

— The RTI was unable to restore.
— One or more joined federates at which the *Federation Restore Begun †* service was invoked have invoked the *Federate Restore Complete* service with a restore-success indicator that indicated failure.
— One or more joined federates at which the *Federation Restore Begun †* service was invoked have resigned.
— The RTI detected failure at one or more of the joined federates at which the *Federation Restore Begun †* service was invoked.
— Federation restore aborted.

All joined federates that received an invocation of the *Federation Restore Begun †* service shall receive an invocation of the *Federation Restored †* service. If a joined federate that received an invocation of the *Federation Restore Begun †* service resigns from the federation execution before the *Federation Restored †* service for that restore is invoked, this resignation shall be considered a failure of the federation restoration, and the *Federation Restored †* service shall be invoked with a restore-success indicator of failure.

4.29.1 Supplied arguments

a) Federation restore-success indicator.
b) Optional failure reason.

4.29.2 Returned arguments

a) None.

4.29.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.

c) The joined federate has a save with the specified label.
d) Federate save not in progress.

4.29.4 Postconditions

a) The joined federate has been informed regarding the success or failure of the restoration attempt.
b) The joined federate may resume providing new information to the federation.

4.29.5 Exceptions

a) Federate internal error.

4.29.6 Reference state charts

a) Figure 3.

4.30 *Abort Federation Restore* service

The *Abort Federation Restore* service requests the RTI to abort the federation restore that is underway.

The service takes no arguments and affects the current restore. This service may be invoked only while the federate is in the Federate Restore in Progress state. Upon return of the service invocation, a request to abort the current restore has successfully been made.

The RTI shall abort the current restore by invoking the *Federation Restored †* service on all federates that are in the Federate Restore in Progress state, regardless of each individual federate's restore substate. Joined federates not in the Federation Restore in Progress state may not receive the *Federation Restore Begun †* callback for this federation restore. The restore-success indicator for the *Federation Restored †* service shall indicate failure (federation restore aborted) if not all federates have successfully invoked the *Federate Restore Complete* service with a restore-success indicator indicating success. Since a federate remains in the Federate Restore in Progress state after it successfully invokes the *Federate Restore Complete* service and before the *Federation Restored †* service is invoked on it by the RTI, it is possible that all other federates may have also already invoked the *Federate Restore Complete* service with a success indicator indicating success. In this case, it is possible that the *Federation Restored †* service is invoked at each federate with a success indicator indicating success after invocation of the *Abort Federation Restore* service.

4.30.1 Supplied arguments

a) None.

4.30.2 Returned arguments

a) None.

4.30.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The federate is in the Federate Restore in Progress state.

4.30.4 Postconditions

a) An abort federation restore has been requested.

4.30.5 Exceptions

a) Federate restore not in progress.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

4.30.6 Reference state charts

a) Figure 3.

4.31 *Query Federation Restore Status* service

The *Query Federation Restore Status* service shall be used to request the status of a federation restore. The RTI shall provide the federation restore status via a *Federation Restore Status Response †* service invocation. No save label argument is needed because at most one federation restore can be in progress.

4.31.1 Supplied arguments

a) None.

4.31.2 Returned arguments

a) None.

4.31.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) Federate save not in progress.

4.31.4 Postconditions

a) The request for a federation restore status has been received by the RTI.

4.31.5 Exceptions

a) Federate save in progress.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

4.31.6 Reference state charts

a) None.

4.32 *Federation Restore Status Response †* service

The *Federation Restore Status Response †* service shall be used to provide the status of a federation restore. This service shall be invoked by the RTI at a joined federate in response to a *Query Federation Restore Status* service invocation by the same joined federate. This service shall supply a list of restore status descriptors. The list shall contain one descriptor for each joined federate participating in the restore. A restore status descriptor shall have the following information:

— Pre-restore joined federate designator
— Post-restore joined federate designator
— Restore status

A joined federate shall use the pre-restore joined federate designators to identify joined federates until the *Initiate Federate Restore †* service has been successfully invoked and indicates restore successful at that joined federate. After that successful service invocation, that joined federate shall use the post-restore joined federate designators to identify joined federates.

A restore status descriptor with a federate status of "No federate restore in progress" shall have a null designator for the post-restore joined federate designator. The post-restore joined federate designator in this case has no meaning since the federate is not participating in a restore.

A joined federate restore status shall be one of the following values:

— No federate restore in progress
— Federate restore request pending
— Federate waiting for restore to begin
— Federate prepared to restore
— Federate restoring
— Federate waiting for federation to restore

4.32.1 Supplied arguments

a) List of joined federates and restore status for each.

4.32.2 Returned arguments

a) None.

4.32.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) Federate save not in progress.
d) Joined federate requested restore status via *Query Federation Restore Status* service.

4.32.4 Postconditions

a) The joined federate has been informed of the status of the federation restore.

4.32.5 Exceptions

a) Federate internal error.

4.32.6 Reference state charts

a) None.

5. Declaration management (DM)

5.1 Overview

Joined federates shall use DM services to declare their intention to generate information. A joined federate shall invoke appropriate DM services before it may register object instances, update instance attribute values, and send interactions. Joined federates shall use DM services or DDM services to declare their intention to receive information. (A joined federate may use DM services exclusively, DDM services exclusively, or both DM and DDM services to declare its intention to receive information. This clause describes how DM services work when they are used exclusively by a joined federate. See 9.1.5 for more information on using DDM services in lieu of or in conjunction with DM services.) A joined federate shall invoke appropriate DM or DDM services before it may discover object instances, reflect instance attribute values, and receive interactions. DM and DDM services, together with object management services, ownership management services, and the object class and interaction class hierarchies defined in the FDD, shall determine the following:

— Object classes at which object instances may be registered
— Object classes at which object instances are discovered
— Instance attributes that are available to be updated and reflected
— Interactions that may be sent
— Interaction classes at which interactions are received
— Parameters that are available to be sent and received

The effects of DM services shall be independent of the logical time of any joined federate in the federation execution.

5.1.1 Static properties of the FDD

The following static properties of the FDD shall establish vocabulary for subsequent DM discussion:

a) Every class shall have at most one immediate superclass. A class shall not be a superclass of a class that is its superclass.
b) Every object class shall have an associated set of class attributes declared in the FDD.
c) An **inherited attribute** of an object class is a class attribute that was declared in a superclass.
d) The **available attributes** of an object class are the set of declared attributes of that object class in union with the set of inherited attributes of that object class.
e) Every interaction class shall have an associated set of parameters declared in the FDD.
f) An **inherited parameter** of an interaction class is a parameter that was declared in a superclass.
g) The **available parameters** of an interaction class are the set of declared parameters of that interaction class in union with the set of inherited parameters of that interaction class.
h) For any service that takes an object class and a set of class attribute designators as arguments, only the available attributes of that object class may be used in the set of attribute designators. Being an available attribute of an object class shall be a necessary, but not necessarily a sufficient, condition for an attribute to be used in the set of attribute designators for such a service.
i) For any service that takes an object instance and a set of attribute designators as arguments, only the available attributes of that object instance's known class at the involved (invoking or invoked) joined federate may be used in the set of attribute designators. Being an available attribute of the object instance's known class shall be a necessary, but not necessarily a sufficient, condition for an attribute to be used in the set of attribute designators for such a service.

5.1.2 Definitions and constraints for object classes and class attributes

The following DM definitions and constraints shall pertain to object classes and class attributes as declared in the class hierarchy of the FDD:

a) A class attribute may be used as an argument to *Subscribe Object Class Attributes*, *Unsubscribe Object Class Attributes, Publish Object Class Attributes*, and *Unpublish Object Class Attributes* service invocations for a particular object class if and only if the attribute is an available attribute of that object class.
b) From a joined federate's perspective, the **subscribed attributes of an object class** shall be all of the class attributes that have been arguments to *Subscribe Object Class Attributes* service invocations by that joined federate for that object class and that have not subsequently been unsubscribed, either individually or by unsubscribing the whole object class. *Subscribe Object Class Attributes* and *Unsubscribe Object Class Attributes* service invocations for one object class shall have no effect on the subscribed attributes of any other object class.
c) If a class attribute is a subscribed attribute of an object class, the joined federate shall be subscribed to that class attribute either actively or passively, but not both.
d) From a joined federate's perspective, the **explicitly published attributes of an object class** shall be all the class attributes that have been arguments to *Publish Object Class Attributes* service invocations by that joined federate for that object class and that have not subsequently been unpublished, either individually or by unpublishing the whole object class.
e) When there are no explicitly published attributes of an object class for a given joined federate, the first *Publish Object Class Attributes* service invocation that results in explicitly published attributes for that object class for that joined federate also results in **publication being established** for that object class at that joined federate.
f) The **HLAprivilegeToDeleteObject** class attribute shall be implicitly published for a joined federate for an object class, if and only if
    1) The **HLAprivilegeToDeleteObject** class attribute for the object class is not, nor has been since publication of the object class was most recently established, explicitly published by the joined federate.
    2) The **HLAprivilegeToDeleteObject** class attribute for the object class has not been explicitly unpublished by the joined federate since publication of the object class was most recently established.
    3) At least one attribute of the object class is explicitly published by the joined federate.
g) From a joined federate's perspective, the published attributes of an object class shall be the explicitly published and the implicitly published, if any, attributes of the object class for the joined federate. *Publish Object Class Attributes* and *Unpublish Object Class Attributes* service invocations for one object class shall have no effect on the published attributes of any other object class.
h) From a joined federate's perspective, an object class shall be **subscribed** if and only if the joined federate is subscribed to at least one attribute of the object class.
i) From a joined federate's perspective, an object class shall be **published** if and only if at least one attribute of the object class is published by the joined federate.
j) Joined federates shall invoke the *Register Object Instance* service only with a published object class as an argument.
k) The **registered class** of an object instance shall be the object class that was an argument to the *Register Object Instance* service invocation for that object instance.
l) Every object instance shall have one federation-wide registered class that cannot change.
m) If the *Discover Object Instance †* service is invoked at a joined federate, the object instance discovered as a result of this service invocation shall have a **discovered class** at that joined federate. The discovered class of the object instance shall be a supplied argument to the *Discover Object Instance †* service invocation.
n) An object instance shall have at most one discovered class in each joined federate. This discovered class may vary from joined federate to joined federate. Once an object instance is discovered, its discovered class shall not change. If a joined federate invokes the *Local Delete Object Instance*

service for a given object instance, that object instance may be newly discovered, at a possibly different object class from previously known.

o) If a joined federate has registered or discovered an object instance and it has not subsequently
   1) Invoked the *Local Delete Object Instance* service for that object instance,
   2) Invoked the *Delete Object Instance* service for that object instance, or
   3) Received an invocation of the *Remove Object Instance †* service for that object instance,

   then the object instance shall be known to that joined federate and that object instance has a known class at that joined federate. The known class of that object instance at that joined federate shall be the object instance's registered class if the joined federate knows about the object instance as a result of having registered it. The known class of that object instance at that joined federate shall be the object instance's discovered class if the joined federate knows about the object instance as a result of having discovered it.

p) From a joined federate's perspective, the corresponding class attribute of an instance attribute is the class attribute of the joined federate's known class for the object instance containing the instance attribute that has the same attribute designator as the instance attribute.

q) From a joined federate's perspective, the corresponding instance attributes of a class attribute are
   1) All unowned instance attributes of object instances that have a known class at the joined federate equal to the object class of the class attribute and that have the same attribute designator as the class attribute, and
   2) All instance attributes owned by the joined federate that belong to object instances that have a known class at the owning federate equal to the object class of the class attribute and that have the same attribute designator as the class attribute.

r) A joined federate shall own and update only an instance attribute for which it is publishing the corresponding class attribute.

s) An update to an instance attribute by the joined federate that owns that instance attribute shall be reflected only by other joined federates that are subscribed to the instance attribute's corresponding (from each subscriber's perspective) class attribute.

5.1.3 Definitions and constraints for interaction classes and parameters

The following DM definitions and constraints shall pertain to interaction classes and parameters as declared in the interaction class hierarchy of the FDD:

a) From a joined federate's perspective, an interaction class shall be **subscribed** if and only if it was an argument to a *Subscribe Interaction Class* service invocation by that joined federate that was not subsequently followed by an *Unsubscribe Interaction Class* service invocation for that interaction class.

b) If an interaction class is subscribed, the joined federate shall be subscribed to that interaction class either actively or passively, but not both.

c) From a joined federate's perspective, an interaction class shall be **published** if and only if it was an argument to a *Publish Interaction Class* service invocation by that joined federate that was not subsequently followed by an *Unpublish Interaction Class* service invocation for that interaction class.

d) Joined federates may invoke the *Send Interaction* service only with a published interaction class as an argument.

e) The sent class of an interaction shall be the interaction class that was an argument to the *Send Interaction* service invocation for that interaction.

f) Every interaction shall have one federation-wide sent class.

g) The *Receive Interaction †* service shall be invoked at a joined federate only with a subscribed interaction class as an argument.

h) If the *Receive Interaction †* service is invoked at a joined federate, the interaction received as a result of this service invocation shall have a **received class** at that joined federate. The received class

of an interaction is the interaction class that is an argument to the *Receive Interaction †* service invocation.

i) An interaction shall have at most one received class in each joined federate. This received class may vary from joined federate to joined federate.
j) Only the available parameters of an interaction class shall be used in a *Send Interaction* service invocation with that interaction class as argument.
k) The sent parameters of an interaction shall be the parameters that were arguments to the *Send Interaction* service invocation for that interaction.
l) The received parameters of an interaction shall be the parameters that were arguments to the *Receive Interaction †* service invocation for that interaction.
m) The received parameters of an interaction shall be the subset of the sent parameters that are available parameters for the interaction's received class.
n) The received parameters for a given interaction may vary from joined federate to joined federate, depending on the received class of the interaction.

When an object instance's discovered class is a superclass of its registered class, the object instance shall be said to have been promoted from the registered class to the discovered class. Similarly, when an interaction's received class is a superclass of its sent class, the interaction shall be said to have been promoted from the sent class to the received class. Promotion is important for protecting federate application code from new subclasses added to the FDD. As the FDD is expanded to include new object and interaction classes that inherit from existing classes, promotion ensures that existing federate application code need not change to work with the expanded FDD.

5.1.4 Declaration management state charts

The following figures depict formal representations of the state of an arbitrary object class, an arbitrary class attribute, the **HLAprivilegeToDeleteObject** class attribute of an arbitrary object class, and an arbitrary interaction class.

Figure 10 depicts the state of an arbitrary object class, and it deals with object classes at two levels. First, it establishes that each class attribute of the object class has some state worth modeling. Second, it establishes that there may be an arbitrary number of instances of each object class. Further, it defines what conditions allow an object instance to be known by a joined federate as an instance of that object class.

Conceptually, the state of an object class comprises the state of the class attributes of that object class and of the object instances of that object class. Furthermore, the state of an object instance comprises the state of the instance attributes of that object instance. There is a correspondence between the instance attributes and their corresponding class attributes. This correspondence is modeled via the index to each attribute. A reference to an instance attribute (*i, k, j*) and to a class attribute (*i, j*) with the same *i*s and same *j*s means that the class attribute is that instance attribute's corresponding class attribute at that joined federate. This correspondence is because having the same *j*s means that they refer to the same attribute designator, and having the same *i*s means that the joined federate's known class for the object instance containing the instance attribute is the same as the object class designator of the class attribute.

Each object class shall have a fixed number of available class attributes as defined in the FDD. The number of object instances of a given class, however, be arbitrary.

An object instance of an object class shall become known by the registering joined federate when the object instance is registered. It may become known by other joined federates in the federation execution. If it becomes known by other joined federates in the federation execution, it shall become known by them as a result of being discovered.

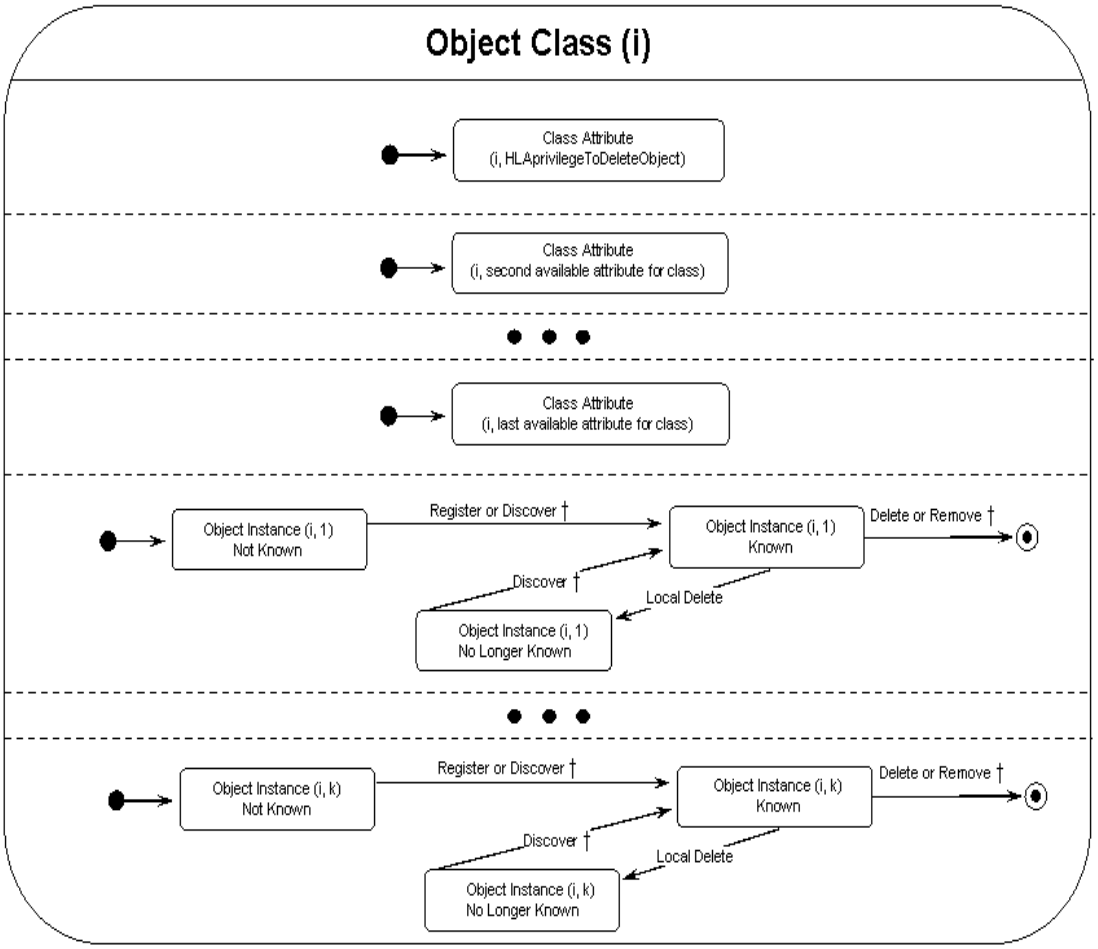

Figure 10—Object class (i)

Figure 11 depicts the state of an arbitrary class attribute (that is not the **HLAprivilegeToDeleteObject** class attribute of an object class) and shows the properties that may be controlled by a joined federate at the class attribute level. Specifically, a joined federate may publish or subscribe to class attributes. Although the *Publish Object Class Attributes* and *Subscribe Object Class Attributes* service invocations can take sets of class attributes as an argument, Figure 11 depicts only a single class attribute. Therefore, for example, *Publish (i, j)* shall mean that attribute *j* of object class *i* was an element of the set used as an argument to the *Publish Object Class Attributes* service.

The **HLAprivilegeToDeleteObject** class attribute of an arbitrary object class is treated slightly differently from other class attributes, and it is depicted in Figure 12. Unlike other class attributes, **HLAprivilegeToDeleteObject**, in addition to explicit publish/unpublish actions, may be implicitly published and unpublished for a given object class at a given federate if other circumstances are met (as shown in the state chart and described in 5.1.2). For purposes of subscription, the **HLAprivilegeToDeleteObject** class attribute of an arbitrary object class is treated no differently from any other class attribute.

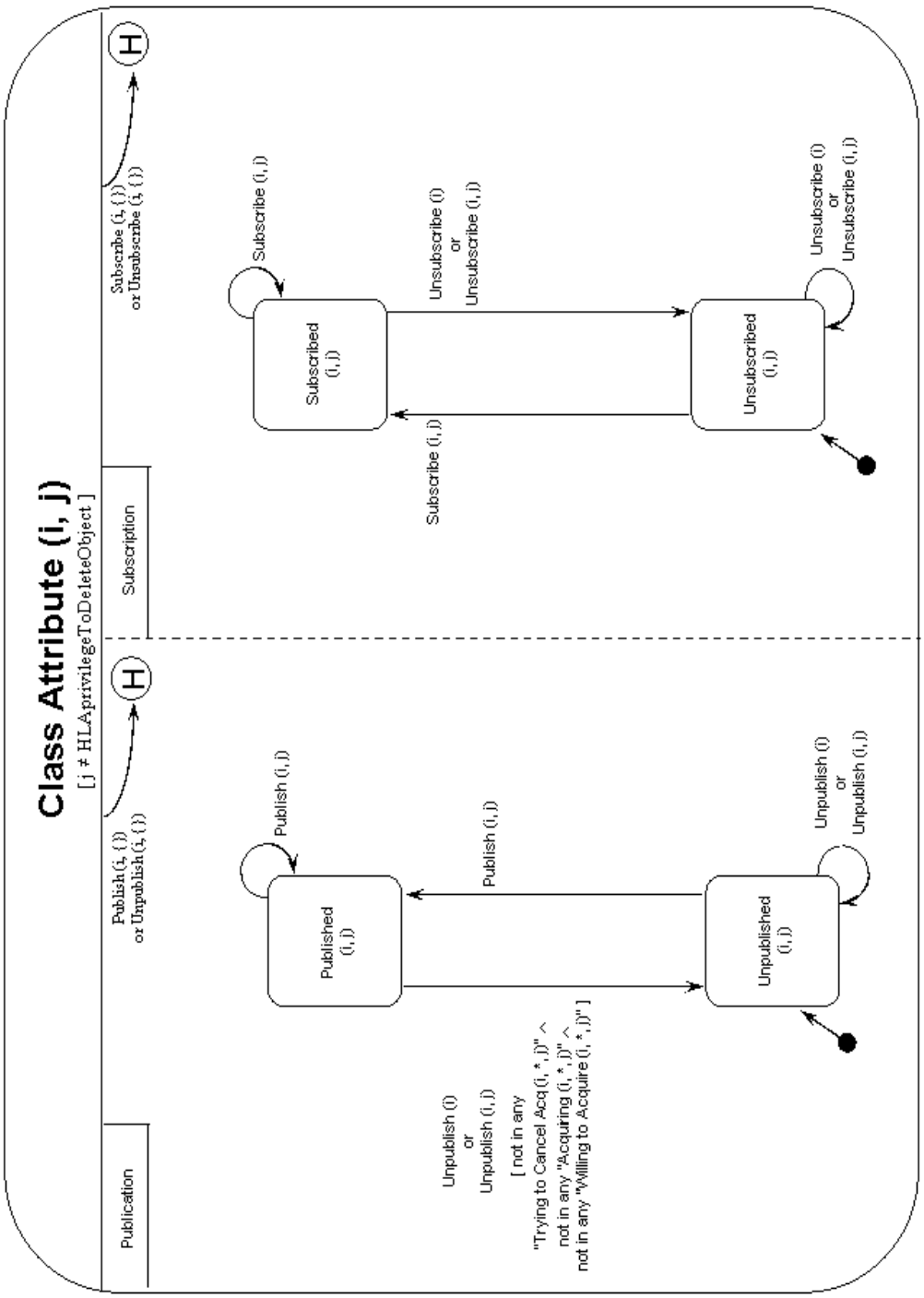

Figure 11—Class attribute (*i*, *j*)

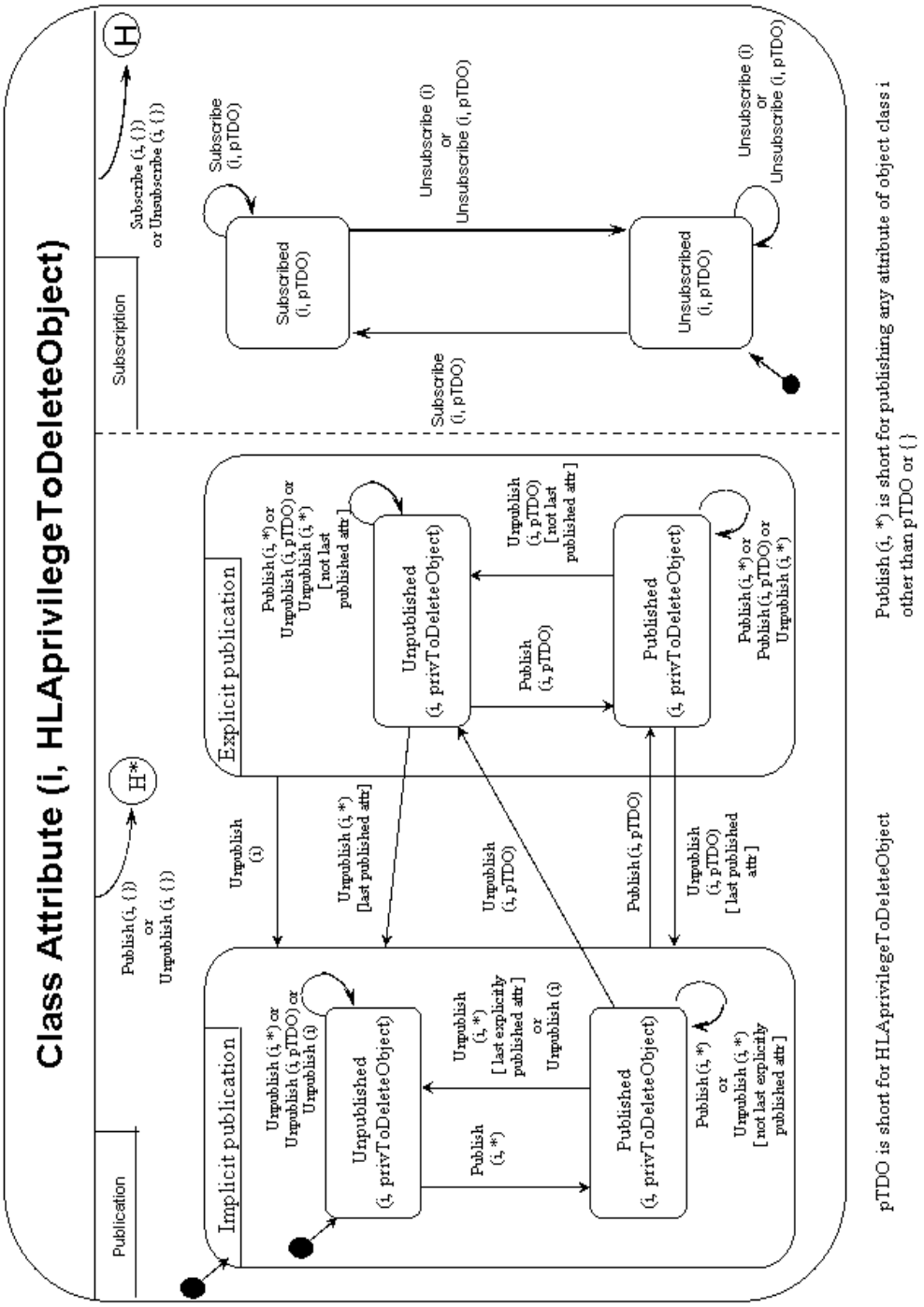

Figure 12—Class attribute (*i*, HLAprivilegeToDeleteObject)

Figure 13 depicts the state of an arbitrary interaction class and shows the properties relating to interaction classes that may be controlled by a joined federate. Specifically, a joined federate may publish or subscribe to interaction classes.

The joined federate may also direct the RTI via the *Enable/Disable Interaction Relevance Advisory Switch* services to indicate that the joined federate does or does not want the RTI to use the *Turn Interactions On †* and *Turn Interactions Off †* services to inform the joined federate when interactions of a given class are relevant to the other joined federates in the federation execution.

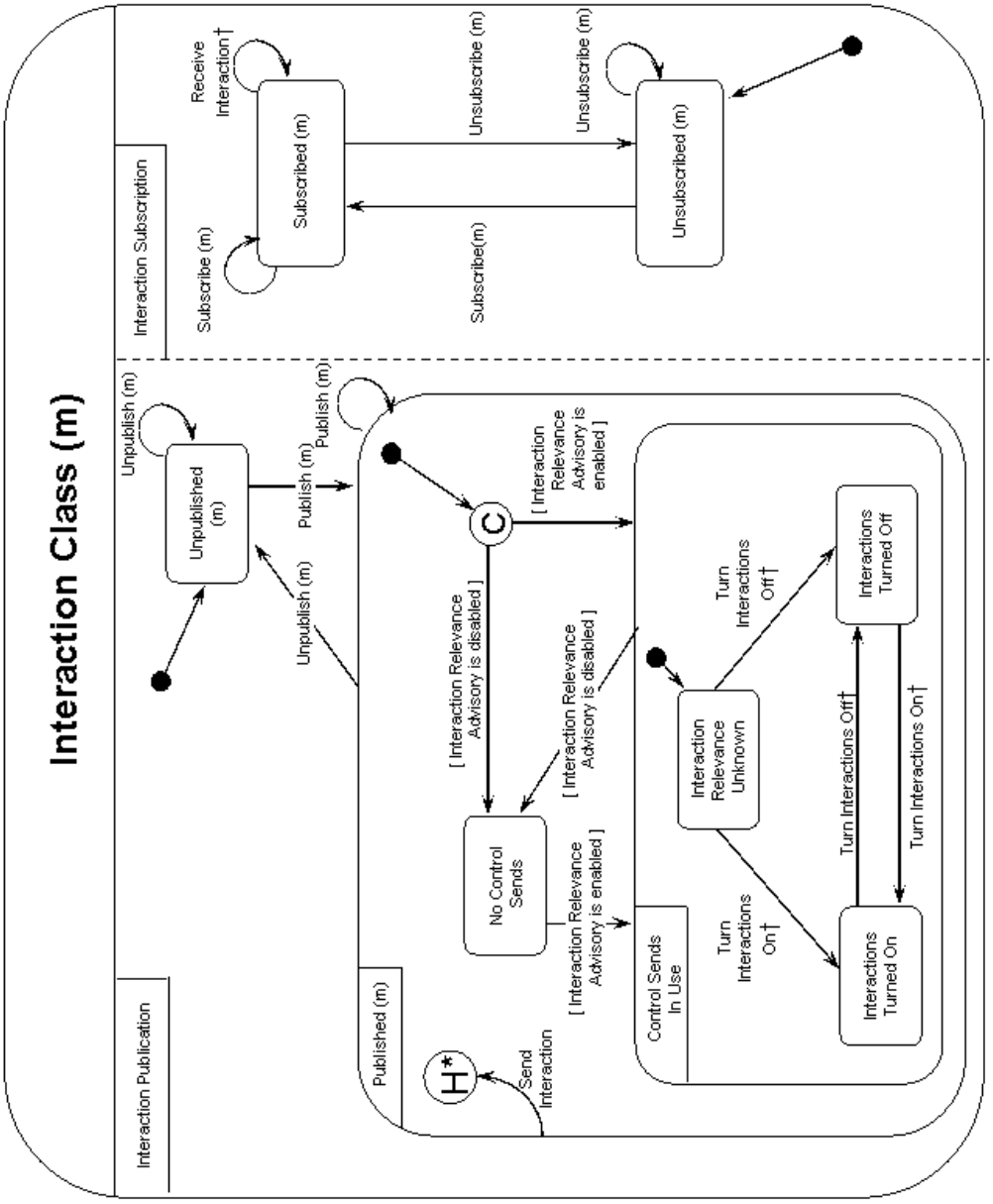

Figure 13—Interaction class (*m*)

5.1.5 Interaction with DDM services

A joined federate may use DM services, and it may also use DDM services. Joined federates that use DM services exclusively may be joined to the same federation execution as joined federates that use DM services exclusively, joined federates that use DDM services exclusively, and joined federates that use both DM services and DDM services. Clause 5 describes how DM services work when they are used in the absence of the use of DDM services by a joined federate, from the perspective of that joined federate, regardless of whether other joined federates in the federation are using DM services exclusively, DDM services exclusively, or both DM services and DDM services. When both DM services and DDM services are used by a single joined federate, some of the terms and services defined in this clause are extended. See 9.1.5 for an expanded interpretation of how selected DM services work when they are used in conjunction with DDM services by a joined federate, from the perspective of that joined federate.

5.1.6 Subscribing with update rate reduction

A federate may, optionally, specify an update rate designator when subscribing to object class attributes. The update rate designator and a corresponding update rate shall be defined in the FDD of the federation execution. If an update rate designator is specified, the attributes shall be considered to be subscribed with the corresponding update rate. In case no update rate is specified, the attributes shall be considered to be subscribed with the default update rate; in other words, no update rate reduction shall take place.

5.2 *Publish Object Class Attributes* service

The information provided by the joined federate via the *Publish Object Class Attributes* service shall be used in multiple ways. First, it shall indicate an object class of which the joined federate may subsequently register object instances. Second, it shall indicate the class attributes of the object class for which the joined federate is capable of owning the corresponding instance attributes. Only the joined federate that owns an instance attribute shall provide values for that instance attribute to the federation. The joined federate may become the owner of an instance attribute and thereby be capable of updating its value in the following two ways:

— By registering an object instance of a published class. Upon registration of an object instance, the registering joined federate shall become the owner of all instance attributes of that object instance for which the joined federate is publishing the corresponding class attributes at the registered class of the object instance.
— By using ownership management services to acquire instance attributes of object instances. The joined federate may acquire only the instance attributes for which the joined federate is publishing the corresponding class attributes at the known class of the specified object instance.

Each use of this service shall add to the publications specified to the RTI in previous service invocations for the same object class. Invoking this service with an empty set of class attributes shall be equivalent to adding no publications for the object class.

5.2.1 Supplied arguments

a) Object class designator.
b) Set of attribute designators.

5.2.2 Returned arguments

a) None.

5.2.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified object class is defined in the FDD.
e) The specified class attributes are available attributes of the specified object class.
f) Federate save not in progress.
g) Federate restore not in progress.

5.2.4 Postconditions

a) The specified class attributes shall now be published attributes of the specified object class for the joined federate.
b) If at least one attribute of the specified class is published by the joined federate, it may register object instances of the specified class.
c) If at least one attribute of the specified class is published by the joined federate, the **HLAprivilegeToDeleteObject** class attribute of the specified class may be implicitly published for the joined federate.
d) The joined federate may own instance attributes that correspond to the specified class attributes.

5.2.5 Exceptions

a) The specified class attributes are not available attributes of the specified object class.
b) The object class is not defined in the FDD.
c) Federate save in progress.
d) Federate restore in progress.
e) The federate is not a federation execution member.
f) Not connected.
g) RTI internal error.

5.2.6 Reference state charts

a) Figure 11.
b) Figure 12.

5.3 *Unpublish Object Class Attributes* service

The *Unpublish Object Class Attributes* service can be used in multiple ways. First, it could be used to unpublish a whole object class and thereby inform the RTI that the joined federate shall no longer be capable of registering object instances of the specified object class. Second, it could be used to unpublish specific class attributes and thereby inform the RTI that the joined federate shall no longer be capable of owning instance attributes that correspond to the unpublished class attributes.

If the optional set of attributes designators argument is supplied, only those class attributes shall be unpublished for the joined federate. Otherwise, all published class attributes of the specified class shall be unpublished for the joined federate. Invoking this service with an empty set of class attributes shall be equivalent to removing no publications for the object class.

The joined federate shall lose ownership of all owned instance attributes whose corresponding class attributes are unpublished by the service invocation; in other words, the joined federate shall no longer be able to update such instance attributes.

5.3.1 Arguments

a) Object class designator.
b) Optional set of attribute designators.

5.3.2 Returned arguments

a) None.

5.3.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The object class is defined in the FDD.
e) If the optional set of attribute designators argument is supplied, the specified class attributes are available attributes of the specified object class.
f) 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 that are unowned by this joined federate and for which the joined federate has taken one of the following actions:
    1) Invoked the *Attribute Ownership Acquisition* service and has not yet received a corresponding invocation of the *Confirm Attribute Ownership Acquisition Cancellation †* service, the *Attribute Ownership Unavailable †* 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 action 1 applies).
g) Federate save not in progress.
h) Federate restore not in progress.

5.3.4 Postconditions

a) If the optional set of attribute designators argument is supplied, the specified attributes are not published by the joined federate for the specified object class.
b) If the optional set of attribute designators argument is supplied and there are no longer any attributes of the class that are explicitly published by the joined federate, the **HLAprivilegeToDeleteObject** class attribute of the specified object class may be implicitly unpublished for the joined federate.
c) If the optional set of attribute designators is not supplied, no class attributes of the object class are published by the joined federate.
d) If there are no class attributes of the specified class that are still published by the joined federate, it shall not be able to register object instances of the specified object class.
e) The joined federate shall no longer own any instance attributes whose corresponding class attributes are no longer published by the joined federate.

5.3.5 Exceptions

a) Cannot unpublish due to pending attempt to acquire instance attribute ownership.
b) The specified class attributes are not available attributes of the specified object class (if the optional set of attribute designators argument is supplied).
c) The object class is not defined in the FDD.
d) Federate save in progress.

e) Federate restore in progress.
f) The federate is not a federation execution member.
g) Not connected.
h) RTI internal error.

5.3.6 Reference state charts

a) Figure 11.
b) Figure 12.

5.4 *Publish Interaction Class* service

The *Publish Interaction Class* service shall inform the RTI of the classes of interactions that the joined federate will send to the federation execution.

5.4.1 Supplied arguments

a) Interaction class designator.

5.4.2 Returned arguments

a) None.

5.4.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The interaction class is specified in the FDD.
e) Federate save not in progress.
f) Federate restore not in progress.

5.4.4 Postconditions

a) The joined federate may now send interactions of the specified class.

5.4.5 Exceptions

a) The interaction class is not defined in the FDD.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

5.4.6 Reference state charts

a) Figure 13.

5.5 *Unpublish Interaction Class* service

The *Unpublish Interaction Class* service shall inform the RTI that the joined federate will no longer send interactions of the specified class.

5.5.1 Supplied arguments

a) Interaction class designator.

5.5.2 Returned arguments

a) None.

5.5.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The interaction class is specified in the FDD.
e) Federate save not in progress.
f) Federate restore not in progress.

5.5.4 Postconditions

a) The joined federate may not send interactions of the specified interaction class.

5.5.5 Exceptions

a) The interaction class is not defined in the FDD.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

5.5.6 Reference state charts

a) Figure 13.

5.6 *Subscribe Object Class Attributes* service

The *Subscribe Object Class Attributes* service shall specify an object class at which the RTI shall notify the joined federate of discovery of object instances. When subscribing to an object class, the joined federate shall also provide a set of class attributes. The values of only the instance attributes that correspond to the specified class attributes, for all object instances discovered as a result of this service invocation, shall be provided to the joined federate from the RTI (via the *Reflect Attribute Values †* service). The set of class attributes provided shall be a subset of the available attributes of the specified object class.

A joined federate shall discover an object instance only as being of a class to which the joined federate is subscribed.

If a joined federate subscribes to multiple locations in an object class inheritance tree, each relevant object instance registration shall result in at most one object instance discovery by the subscribing joined federate.

The discovered class shall be the registered class if subscribed by the discovering joined federate. Otherwise, the discovered class shall be the closest superclass of the registered class subscribed by the discovering joined federate.

Each use of this service shall add to the subscriptions specified to the RTI in any previous *Subscribe Object Class Attributes* service invocation for the same object class. Invoking this service with an empty set of class attributes shall be equivalent to adding no subscriptions for the specified object class.

The use of the optional passive subscription indicator shall act 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 shall 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.

If the optional passive subscription indicator argument indicates that this is a passive subscription:

a) The invocation of this service shall not cause the *Start Registration For Object Class †* service to be invoked at any other joined federate, and
b) If this invocation replaces a previous subscription that was active rather than passive, invocation of this service may cause the *Stop Registration For Object Class †* service to be invoked at one or more other joined federates, depending on the joined federate's publishing status.

If the optional passive subscription indicator argument is not present or indicates that this is an active subscription, the invocation of this service may cause the *Start Registration For Object Class* service to be invoked at one or more other joined federates.

If the optional maximum update rate is specified, the attributes shall be considered to be subscribed with the corresponding update rate. If no update rate is specified, the attributes shall be considered to be subscribed with the default update rate; in other words, no update rate reduction shall take place.

5.6.1 Supplied arguments

a) Object class designator.
b) Set of attribute designators.
c) Optional passive subscription indicator.
d) Optional update rate designator.

5.6.2 Returned arguments

a) None.

5.6.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified object class is defined in the FDD.

e) The specified class attributes are available attributes of the specified object class.
f) Federate save not in progress.
g) Federate restore not in progress.

5.6.4 Postconditions

a) The RTI has been informed of the joined federate's requested subscription.

5.6.5 Exceptions

a) The specified class attributes are not available attributes of the specified object class.
b) The object class is not defined in the FDD.
c) Invalid update rate designator.
d) Federate save in progress.
e) Federate restore in progress.
f) The federate is not a federation execution member.
g) Not connected.
h) RTI internal error.

5.6.6 Reference state charts

a) Figure 11.
b) Figure 12.

5.7 *Unsubscribe Object Class Attributes* service

The *Unsubscribe Object Class Attributes* service can be used in multiple ways. First, it could be used to unsubscribe a whole class and thereby inform the RTI that it shall stop notifying the joined federate of object instance discovery at the specified object class. Second, it could be used to unsubscribe specific class attributes and thereby inform the RTI that the joined federate shall no longer receive values for any of the unsubscribed class attributes corresponding instance attributes.

If the optional set of attribute designators argument is supplied, only those class attributes shall be unsubscribed for the joined federate. Otherwise, all subscribed class attributes of the specified class shall be unsubscribed for the joined federate. Invoking this service with an empty set of class attributes shall be equivalent to removing no subscriptions for the object class.

All in-scope instance attributes whose corresponding class attributes are unsubscribed by the service invocation shall go out of scope. Refer to 9.1.1 for an expanded interpretation of this service when a joined federate is using DDM services in conjunction with DM services.

Invocation of this service, subject to other conditions, may cause an implicit out-of-scope situation for affected instance attributes, i.e., invoking joined federate shall NOT be notified via an invocation of the *Attributes Out Of Scope †* service invocation when the instance attribute goes out of scope.

5.7.1 Supplied arguments

a) Object class designator.
b) Optional set of attribute designators.

5.7.2 Returned arguments

a) None.

5.7.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The object class is defined in the FDD.
e) If the optional set of attribute designators argument is supplied, the specified class attributes are available attributes of the specified object class.
f) Federate save not in progress.
g) Federate restore not in progress.

5.7.4 Postconditions

a) If the optional set of attribute designators argument is supplied, the specified attributes are not subscribed by the joined federate.
b) If the optional set of attribute designators argument is not supplied, no class attributes of the object class are subscribed by the joined federate.
c) If no class attributes of the specified class are still subscribed by the joined federate, the joined federate shall receive no subsequent *Discover Object Instance* service invocations for the specified object class.
d) The joined federate shall receive no subsequent *Reflect Attribute Values †* service invocations for any instance attributes whose corresponding class attributes are no longer subscribed by the joined federate.
e) May cause an implicit out-of-scope situation for affected instance attributes at invoking joined federate.

5.7.5 Exceptions

a) The specified class attributes are not available attributes of the specified object class (if the optional set of attribute designators argument is supplied).
b) The object class is not defined in the FDD.
c) Federate save in progress.
d) Federate restore in progress.
e) The federate is not a federation execution member.
f) Not connected.
g) RTI internal error.

5.7.6 Reference state charts

a) Figure 11.
b) Figure 12.

5.8 *Subscribe Interaction Class* service

The *Subscribe Interaction Class* service specifies an interaction class for which the RTI shall notify the joined federate of sent interactions by invoking the *Receive Interaction †* service at the joined federate.

When an interaction is received by a joined federate, the received class of the interaction shall be the interaction's sent class, if subscribed. Otherwise, the received class is the closest superclass of the sent class that is subscribed when the interaction is received. Only the parameters from the interaction's received class and its superclasses will be received.

If a joined federate subscribes to multiple locations in an interaction class inheritance tree, each relevant interaction sent shall result in at most one received interaction in the subscribing joined federate.

If the optional passive subscription indicator indicates that this is a passive subscription state:

a) The invocation of this service shall not cause the *Turn Interactions On †* service to be invoked at any other joined federate, and
b) If this invocation replaces a previous subscription that was active rather than passive, invocation of this service may cause the *Turn Interactions Off †* service to be invoked at one or more other joined federates, depending on the joined federate's publishing status.

If the optional passive subscription indicator is not present or indicates that this is an active subscription, the invocation of this service may cause the *Turn Interactions On †* service to be invoked at one or more other joined federates.

If a joined federate has the Service Reporting Switch Enabled, that joined federate shall not be able to subscribe to the HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation interaction class.

5.8.1 Supplied arguments

a) Interaction class designator.
b) Optional passive subscription indicator.

5.8.2 Returned arguments

a) None.

5.8.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The interaction class is defined in the FDD.
e) If the specified interaction class designator is the MOM interaction class HLAmanager. HLAfederate.HLAreport.HLAreportServiceInvocation, the federate's service invocations are not being reported via MOM interactions.
f) Federate save not in progress.
g) Federate restore not in progress.

5.8.4 Postconditions

a) The RTI has been informed of the joined federate's requested subscription.

5.8.5 Exceptions

a) Federate service invocations are being reported via MOM interactions.
b) The interaction class is not defined in the FDD.
c) Federate save in progress.
d) Federate restore in progress.
e) The federate is not a federation execution member.
f) Not connected.
g) RTI internal error.

5.8.6 Reference state charts

a) Figure 13.

5.9 *Unsubscribe Interaction Class* service

The *Unsubscribe Interaction Class* service informs the RTI that it shall no longer notify the joined federate of sent interactions of the specified interaction class. See 9.1.5 for an expanded interpretation of this service when DDM is used.

5.9.1 Supplied arguments

a) Interaction class designator.

5.9.2 Returned arguments

a) None.

5.9.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The interaction class is defined in the FDD.
e) Federate save not in progress.
f) Federate restore not in progress.

5.9.4 Postconditions

a) The RTI shall not deliver interactions of the specified interaction class to the joined federate.

5.9.5 Exceptions

a) The interaction class is not defined in the FDD.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

5.9.6 Reference state charts

a) Figure 13.

5.10 *Start Registration For Object Class †* service

The *Start Registration For Object Class †* service shall notify the joined federate that registration of new object instances of the specified object class is advised because at least one of the class attributes that the joined federate is publishing at this object class is actively subscribed to at the specified object class or at a superclass of the specified object class by at least one other joined federate in the federation execution. The joined federate should commence with registration of object instances of the specified class. Generation of the *Start Registration For Object Class †* service advisory shall be controlled using the *Enable/Disable*

Object Class Relevance Advisory Switch services. The *Start Registration For Object Class †* service shall be invoked only when the Object Class Relevance Advisory Switch is enabled for the joined federate.

5.10.1 Supplied arguments

a) Object class designator.

5.10.2 Returned arguments

a) None.

5.10.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) 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.
d) The Object Class Relevance Advisory Switch is enabled for the joined federate.
e) Federate save not in progress.
f) Federate restore not in progress.

5.10.4 Postconditions

a) The joined federate has been notified of the requirement to begin registering object instances of the specified object class.

5.10.5 Exceptions

a) Federate internal error.

5.10.6 Reference state charts

a) None.

5.11 *Stop Registration For Object Class †* service

The *Stop Registration For Object Class †* service shall notify the joined federate that registration of new object instances of the specified object class is not advised because none of the class attributes that the joined federate is publishing at this object class is actively subscribed to at the specified object class or at a superclass of the specified object class by any other joined federate in the federation execution. The joined federate should stop registration of new object instances of the specified class. Generation of the *Stop Registration For Object Class †* service advisory shall be controlled using the *Enable/Disable Object Class Relevance Advisory Switch* services. The *Stop Registration For Object Class †* service shall be invoked only when the Object Class Relevance Advisory Switch is enabled for the joined federate.

5.11.1 Supplied arguments

a) Object class designator.

5.11.2 Returned arguments

a) None.

5.11.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate is publishing the object class.
d) 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.
e) The Object Class Relevance Advisory Switch is enabled for the joined federate.
f) Federate save not in progress.
g) Federate restore not in progress.

5.11.4 Postconditions

a) The joined federate has been notified of the requirement to stop registration of object instances of the specified object class.

5.11.5 Exceptions

a) Federate internal error.

5.11.6 Reference state charts

a) None.

5.12 *Turn Interactions On †* service

The *Turn Interactions On †* service shall notify the joined federate that the specified class of interactions is relevant because it or a superclass is actively subscribed to by at least one other joined federate in the federation execution. The joined federate should commence with the federation-agreed-upon scheme for sending interactions of the specified class. Generation of the *Turn Interactions On †* service advisory shall be controlled using the *Enable/Disable Interaction Relevance Advisory Switch* services. The *Turn Interactions On †* service shall be invoked only when the Interaction Relevance Advisory Switch is enabled for the joined federate.

5.12.1 Supplied arguments

a) Interaction class designator.

5.12.2 Returned arguments

a) None.

5.12.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate is publishing the interaction class.

d) Some other joined federate is actively subscribed to the interaction class or to a superclass of the interaction class.
e) The Interaction Relevance Advisory Switch is enabled for the joined federate.
f) Federate save not in progress.
g) Federate restore not in progress.

5.12.4 Postconditions

a) The joined federate has been notified that some other joined federate in the federation execution is subscribed to the interaction class.

5.12.5 Exceptions

a) Federate internal error.

5.12.6 Reference state charts

a) Figure 13.

5.13 *Turn Interactions Off †* service

The *Turn Interactions Off †* service shall indicate to the joined federate that the specified class of interactions is not relevant because it or a superclass is not actively subscribed to by any other joined federate in the federation execution. Generation of the *Turn Interactions Off †* service advisory shall be controlled using the *Enable/Disable Interaction Relevance Advisory Switch* services. The *Turn Interactions Off †* service shall be invoked only when the Interaction Relevance Advisory Switch is enabled for the joined federate.

5.13.1 Supplied arguments

a) Interaction class designator.

5.13.2 Returned arguments

a) None.

5.13.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate is publishing the interaction class.
d) No other joined federate is actively subscribed to the interaction class or to a superclass of the interaction class.
e) The Interaction Relevance Advisory Switch is enabled for the joined federate.
f) Federate save not in progress.
g) Federate restore not in progress.

5.13.4 Postconditions

a) The joined federate has been notified that no other joined federate in the federation execution is subscribed to the interaction class.

5.13.5 Exceptions

a) Federate internal error.

5.13.6 Reference state charts

a) Figure 13.

6. Object management

6.1 Overview

This group of HLA services shall deal with the registration, modification, and deletion of object instances and the sending and receipt of interactions.

6.1.1 Object instance discovery

Object instance discovery is a prime concept in this service group. Object instance *O* shall have a candidate discovery class at joined federate *F* if joined federate *F* is subscribed to either the registered class of *O* or to a superclass of the registered class of *O*. (A joined federate *F* may be subscribed either by the DM subscription service *Subscribe Object Class Attributes* or by the DDM subscription service *Subscribe Object Class Attributes With Regions*.) If an object instance has a candidate discovery class at a joined federate, the candidate discovery class of the object instance at that joined federate shall be the object instance's registered class if subscribed to by the joined federate. Otherwise, the candidate discovery class of the object instance shall be the closest superclass of the object instances's registered class to which the joined federate is subscribed.

A joined federate discovers an object instance via the *Discover Object Instance †* service. This service shall be invoked at a joined federate *F* for object instance *O* when

a) *O* is not known at *F,* and
b) There is an instance attribute *i* of *O* that has a corresponding class attribute *i′,* and
c) *i* is owned by either another joined federate (not *F*) or the RTI, and
d) Either
    1) *i′* is a subscribed attribute of *O*'s candidate discovery class, or
    2) *i′* is a subscribed attribute of *O*'s candidate discovery class with regions and the update region set of *i* by the owning joined federate overlaps the subscription region set of *i′* at *O*'s candidate discovery class at the subscribing joined federate

When the *Discover Object Instance †* service is invoked, the class that is an argument to this service invocation shall be called the discovered class of the object instance. At the moment of discovery, the discovered class shall be the same as the candidate discovery class. Subsequent to discovery, the discovered class cannot change. The candidate discovery class may change. As long as an object instance remains known, however, its candidate discovery class is not of interest.

When a joined federate either uses the *Register Object Instance* service to register an object instance or receives an invocation of the *Discover Object Instance †* to discover an object instance, that object instance becomes known to the joined federate, and the object instance has a known class at that joined federate. If a joined federate knows about an object instance as a result of having registered it, that object instance's known class is its registered class. If the joined federate knows about the object instance as a result of having discovered it, the object instance's known class is its discovered class.

6.1.2 Scope

When the *Discover Object Instance †* service is invoked, there shall be an instance attribute that is part of the newly discovered object instance that immediately comes into scope at the discovering joined federate, both when DDM is used and when it is not used. 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 or by the RTI, and

c) Either
   1) The instance attribute's corresponding class attribute is a subscribed attribute of the known class of the object instance, or
   2) 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.

6.1.3 In-scope reflects

If an instance attribute is in scope for a given joined federate ***F***, exactly one other joined federate exists (***U***) in the federation execution that is capable of invoking the *Update Attribute Values* service for that instance attribute so that if ***U*** were to invoke the *Update Attribute Values* service for that instance attribute, all of the preconditions for the invocation of the corresponding *Reflect Attribute Values †* service at ***F*** are satisfied.

The *Reflect Attribute Values †* service may be invoked at a joined federate for a given instance attribute when

a) The instance attribute is in scope.
b) A corresponding *Update Attribute Values* service for the instance attribute was invoked by another federate.

6.1.4 Out-of-scope reflects

The *Reflect Attribute Values †* service may also be invoked at a joined federate for a given instance attribute when

a) The instance attribute is out of scope.
b) A corresponding *Update Attribute Values* service for the instance attribute was invoked by another federate when the instance attribute was in scope.

6.1.5 Attribute relevance

A joined federate may also direct the RTI, via the *Enable/Disable Attribute Relevance Advisory Switch* services, to indicate that the joined federate does or does not want the RTI to use the *Turn Updates On For Object Instance †* and *Turn Updates Off For Object Instance †* services to inform the joined federate when updates to particular instance attributes are relevant to the other joined federates in the federation execution.

6.1.6 Orphan object instances

Note that there are obscure cases under which an object instance shall become orphaned within a federation execution. Orphaned object instances are unknown by all joined federates and cannot be discovered by any means. An object instance shall become orphaned only if no joined federate knows the object instance (i.e., because the last joined federate to know the object instance has resigned or invoked *Local Delete Object Instance*). A consequence of no joined federate knowing the object instance is that no instance attributes of that object instance shall be owned. Because the definition of discovery requires that at least one instance attribute be owned, no joined federates can discover the object instance. Without a joined federate being able to discover the object instance, no joined federate can acquire ownership of an instance attribute via the ownership management services. Orphaned object instances shall be unreachable by joined federates and shall be dealt with appropriately by the RTI.

6.1.7 Interactions

Interaction receipt is also an important concept in the object management service group. Interaction *I* shall have a candidate received class at joined federate ***F*** if joined federate ***F*** is subscribed to either the sent class

of *I* or to a superclass of the sent class of *I*. (A joined federate ***F*** may be subscribed to an interaction class either by the DM subscription service *Subscribe Interaction Class* or by the DDM subscription service *Subscribe Interaction Class With Regions*.) If an interaction has a candidate received class at a joined federate, the candidate received class of the interaction at that joined federate shall be the interaction's sent class if subscribed to by the joined federate. Otherwise, the candidate received class of the interaction shall be the closest superclass of the interaction's sent class to which the joined federate is subscribed.

A joined federate receives an interaction via the *Receive Interaction †* service. This service may be invoked at a joined federate ***F*** when

a) Another joined federate (not *F*) has invoked the *Send Interaction* service to send interaction *I*, and
b) *I* has a candidate received class at *F,* and
c) Either
    1) This candidate received class is a subscribed interaction class, or
    2) This candidate received class is a subscribed interaction class with regions, and the update region set of *I* overlaps the subscription region set for *I*'s candidate received class at the subscribing joined federate.

When the *Receive Interaction †* service is invoked, the class that is an argument to this service invocation shall be called the received class of the interaction that is received as a result of this service invocation. At the moment of receipt, the received class shall be the same as the candidate received class.

6.1.8 State charts

Figure 14 depicts formal representations of the state of an arbitrary object instance.

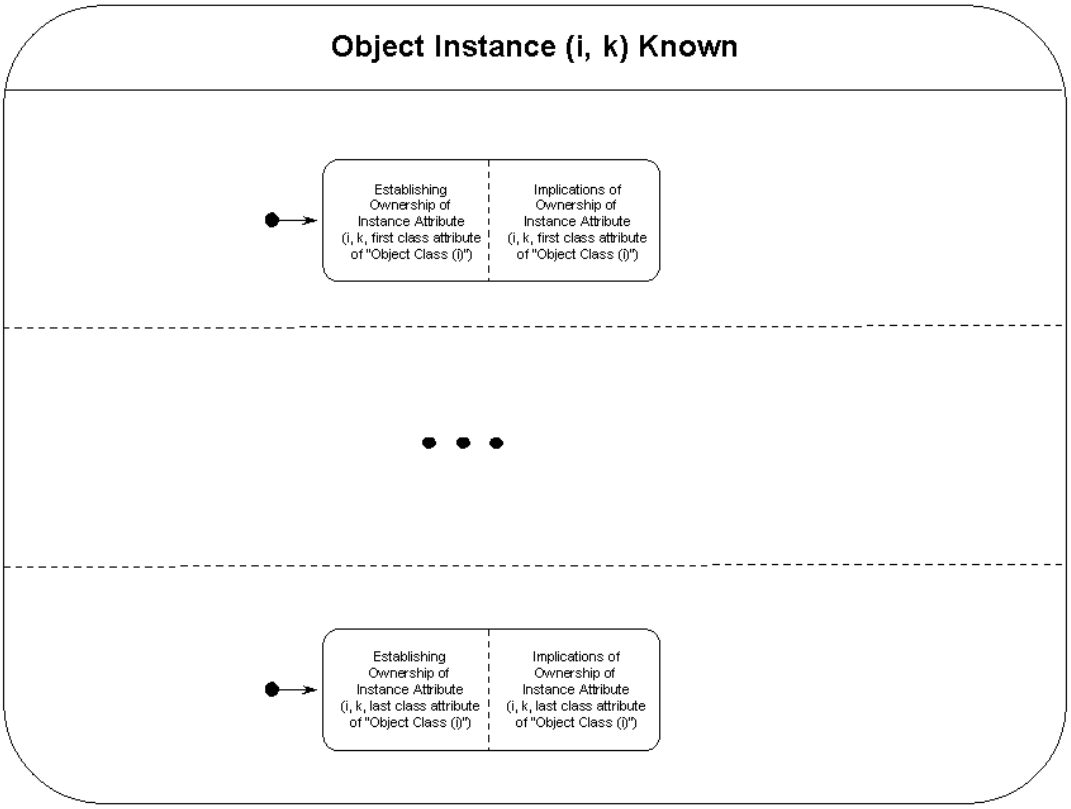

Figure 14—Object instance (*i*, *k*) known

Figure 15 depicts the implications of ownership of an arbitrary instance attribute.

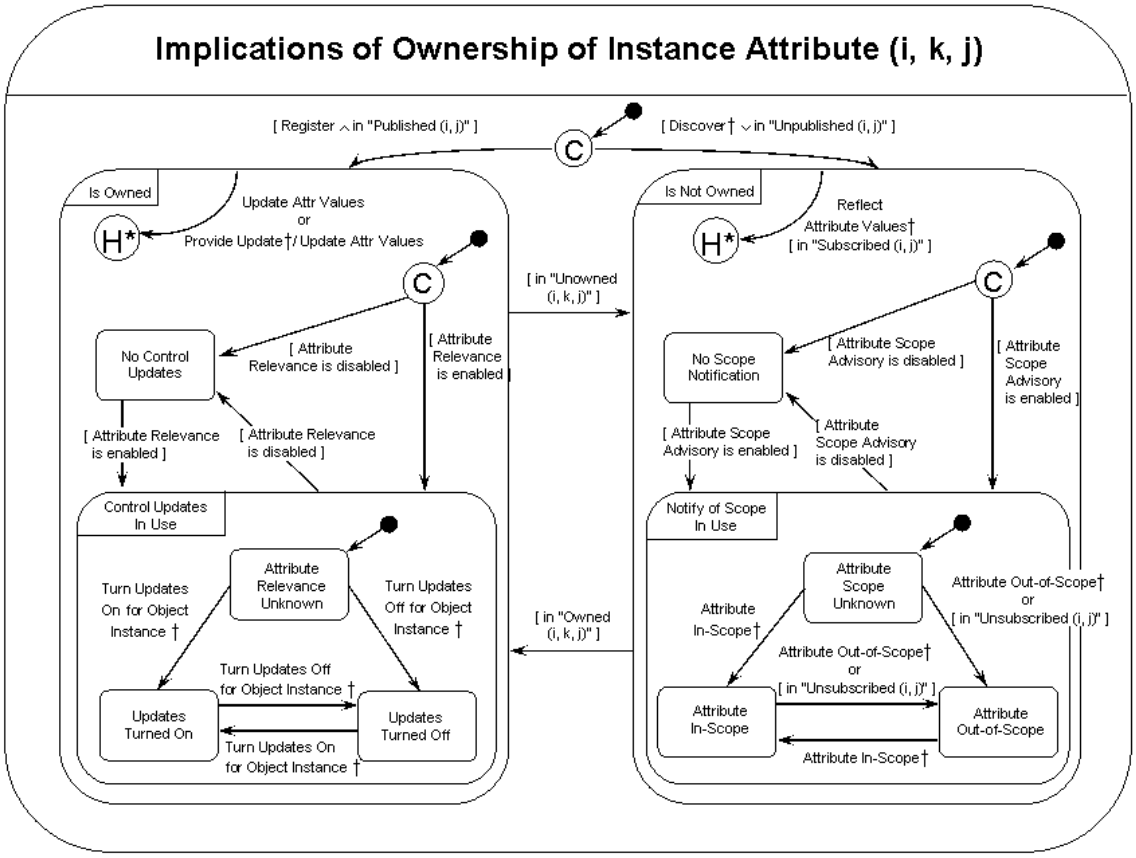

Figure 15—Implications of ownership of instance attribute (*i*, *k*, *j*)

6.1.9 Auto-Provide Switch

The federation execution-wide Auto-Provide Switch indicates whether the RTI shall automatically solicit attribute value updates from attribute-owning federates. The initial setting for this switch shall come from the FDD supplied to the *Create Federation Execution* service invocation for the federation execution. This switch setting shall be modifiable during a federation execution via the HLAmanager.HLAfederation.HLA-adjust.HLAsetSwitches MOM interaction.

Whenever this switch is enabled, the *Provide Attribute Value Update †* service shall be invoked, at an instance attribute-owning federate, for every instance attribute in an execution, whenever it becomes in scope at another non-owning federate. If an instance attribute becomes in scope at multiple federates at the same wall-clock time, the *Provide Attribute Value Update †* service may be invoked at the owning federate just once, rather than once for each federate at which the instance attribute became in scope.

When this switch changes to enabled during a federation execution, the *Provide Attribute Value Update †* service shall be invoked at the appropriate owning federate for every in-scope instance attribute.

This switch relies on the definition of in-scope attributes, not on the setting of the Attribute Scope Advisory Switch. The Auto-Provide Switch shall function the same whether the Attribute Scope Advisory Switch is enabled or disabled.

6.1.10 Transportation types

The type of transportation that carries instance attribute values and interactions can be controlled using various HLA facilities. A transportation type shall be defined as either reliable (which means that delivery of one copy of the data is guaranteed) or best effort (which means that data may be lost in transport). The following transportation types shall always be supported by an RTI:

a) HLAreliable: which is the default reliable transportation type
b) HLAbestEffort: which is the default best-effort transportation.

Other transportation types may be supported by the RTI. If a transportation type is requested by a joined federate or in the FDD but not supported by the RTI or other infrastructure components, the RTI may instead fall back to a corresponding (reliable or best-effort) transportation type of its choosing. Ultimately the RTI shall fallback to the corresponding default transportation type (HLAreliable or HLAbestEffort).

A joined federate may request a change of the transportation type for attributes that it owns using the *Request Attribute Transportation Type Change* service. When this operation has completed, the federate shall receive an invocation of the *Confirm Attribute Transportation Type Change †* service, specifying the transportation type that was actually achieved, which may or may not have changed.

Similarly, a joined federate may request a change of the transportation type for interactions using the *Request Interaction Transportation Type Change* service. When this operation has completed, the joined federate shall receive an invocation of the *Confirm Interaction Transportation Type Change †* service, specifying the transportation type that was actually achieved, which may or may not have changed.

In case the FDD specifies a transportation type that is not available, the RTI shall silently fallback to a corresponding transportation type. It is still possible for a joined federate to inspect the transportation type employed by using the *Query Attribute Transportation Type* service, which will result in a *Report Attribute Transportation Type †* service invocation. The corresponding services for interactions are *Query Interaction Transportation Type* and *Report Interaction Transportation Type †*.

6.1.11 Passelization

All of the instance attribute/value pairs in an *Update Attribute Values* service invocation for instance attributes that have identical transportation type, sent message order type, and set of sent region designators (if present) shall be defined as one passel. The instance attributes that make up a passel shall be available for delivery in the same *Reflect Attribute Values* service invocation. The delivery of the instance attributes that make up the passel shall depend on the subscriptions and subscription region specifications at the reflecting joined federate. Elements shall not change passels, passels shall not be divided, and passels shall not be combined with other passels. Since one *Update Attribute Values* service invocation may result in the creation of multiple passels (where sent transportation type, sent message order type, and set of sent region designators are different), an *Update Attribute Values* service invocation may result in multiple *Reflect Attribute Values* service invocations in a reflecting joined federate. Instance attributes delivered via a *Reflect Attribute Values* service invocation shall not contain elements from more than one *Update Attribute Values* service invocation.

6.1.12 Sending updates with update rate reduction

The RTI shall not provide an update rate to a joined federate that exceeds the subscribed update rate. It shall not, however, deliver any more updates than what is provided by the owning joined federate. In other words, any minimum update rate shall not be guaranteed. In case a joined federate has subscribed with an update rate other than the default update rate, the RTI shall handle this situation as follows:

a) for attributes with best-effort transportation, messages shall not be delivered to a joined federate with an update rate higher than the subscribed update rate. Any messages in excess of this rate shall be dropped. Given a maximum update rate of x Hz and given that a message has been delivered at wall-clock time t, no message shall be delivered before $t + (1 / x)$.
b) for attributes with reliable transportation, no messages shall be dropped.

In case the attribute relevancy advisory switch is enabled, the owning joined federate shall be notified of the maximum actively subscribed update rate as part of the *Turn Updates On For Object Instance †* service invocation. The *Turn Updates On for Object Instance †* service invocation shall also be invoked at owning joined federates whenever the maximum actively subscribed update rate in the federation execution changes. In case any joined federate is actively subscribed using the default update rate, no update rate designator will be provided as part of the *Turn Updates On for Object Instance †* service invocation. In case the transportation type for an attribute is changed, the update rate reduction behavior for the new transportation type shall apply.

If operating in a synchronous callback model, a subscribing joined federate whose rate of invoking the *Evoke Callback* service (*Re*) and rate of updates by the publishing joined federate (*Rp*) are higher than the subscribed update rate (*Rs*), then the RTI shall drop messages at a rate of *Rp–Rs*. If either *Re* or *Rp* is less than *Rs*, there are no guarantees on the rate of messages received by the subscribing joined federate.

6.2 *Reserve Object Instance Name* service

The *Reserve Object Instance Name* service indicates that the joined federate is attempting to reserve the exclusive right to register object instances with the supplied name (subject to the restriction that only one object instance at a time shall exist with a particular name). The value of the supplied name argument shall not begin with "HLA." The RTI shall provide the result of the reservation attempt via the *Object Instance Name Reserved †* service.

6.2.1 Supplied arguments

a) Name.

6.2.2 Returned arguments

a) None.

6.2.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The value of the name argument does not begin with "HLA."
e) Federate save not in progress.
f) Federate restore not in progress.

6.2.4 Postconditions

a) The RTI has the candidate name.

6.2.5 Exceptions

a) Illegal name.
b) Federate save in progress.

c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

6.2.6 Reference state charts

a) None.

6.3 *Object Instance Name Reserved †* service

The *Object Instance Name Reserved †* service notifies the joined federate whether the name provided in a previous invocation of *Reserve Object Instance Name* service has been reserved. A reservation-success indicator argument indicating success shall mean that the joined federate has obtained the unique right to register object instances with the supplied name. The reservation is retained until the joined federate either resigns or relinquishes the reservation by successfully invoking the *Release Object Instance Name* service. While a joined federate holds a reservation on a name, no other federate joined to the current federation execution shall receive a successful reservation for that name.

6.3.1 Supplied arguments

a) Name.
b) Reservation-success indicator.

6.3.2 Returned arguments

a) None.

6.3.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate had previously invoked the *Reserve Object Instance Name* service with the name.
d) Federate save not in progress.
e) Federate restore not in progress.

6.3.4 Postconditions

a) If the reservation-success indicator is true, the joined federate has been notified that it has obtained the unique right to register objects with the supplied name.

6.3.5 Exceptions

a) Federate internal error.

6.3.6 Reference state charts

a) None.

6.4 *Release Object Instance Name* service

The *Release Object Instance Name* service indicates that the joined federate is attempting to relinquish the unique right to register object instances with the supplied name. The supplied name shall be a name that is currently reserved by the joined federate, having previously been the subject of a successful invocation of the *Object Instance Name Reserved †* service. Once an object instance name reservation is released, the name shall be available for any joined federate to reserve.

6.4.1 Supplied arguments

a) Name.

6.4.2 Returned arguments

a) None.

6.4.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The supplied name is currently reserved by the joined federate, having previously been the subject of a successful invocation of either the *Object Instance Name Reserved †* service or the *Multiple Object Instance Names Reserved †* service.
e) Federate save not in progress.
f) Federate restore not in progress.

6.4.4 Postconditions

a) The supplied name is no longer reserved by the joined federate.

6.4.5 Exceptions

a) The object instance name is not currently reserved.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

6.4.6 Reference state charts

a) None.

6.5 *Reserve Multiple Object Instance Names* service

The *Reserve Multiple Object Instance Names* service indicates that the joined federate is attempting to reserve the exclusive right to register object instances with the supplied names (subject to the restriction that only one object instance at a time shall exist with a particular name). The name set cannot be empty. The name values in the supplied name set argument shall not begin with "HLA." Any one illegal name shall abort the reservation of all names in the set. The RTI shall provide the result of the reservation attempt via the *Multiple Object Instance Name Reserved †* service. The *Object Instance Name Reserved †* service shall not be used to provide the result for any single name.

6.5.1 Supplied arguments

a) Name Set.

6.5.2 Returned arguments

a) None.

6.5.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The name set argument is not empty.
e) Each name value in the name set argument does not begin with "HLA."
f) Save not in progress.
g) Restore not in progress.

6.5.4 Postconditions

a) The RTI has the candidate names.

6.5.5 Exceptions

a) The name set was empty.
b) Illegal name.
c) The federate is not a federation execution member.
d) Save in progress.
e) Restore in progress.
f) Not connected.
g) RTI internal error.

6.5.6 Reference state charts

a) None.

6.6 *Multiple Object Instance Names Reserved †* service

The *Multiple Object Instance Names Reserved* † service notifies the joined federate whether the names provided in a previous invocation of *Reserve Multiple Object Instance Names* service have been reserved. If successful, this reservation shall mean that the joined federate has obtained the unique right to register object instances with the supplied names. The reservation is retained until the joined federate either resigns or relinquishes the reservation by successfully invoking the *Release Object Instance Name* or *Release Multiple Object Instance Names* service. While a joined federate holds a reservation on a name, no other federate joined to the current federation execution shall receive a successful reservation for that name. This service shall not be invoked as a result of the *Reserve Object Instance Name* service.

6.6.1 Supplied arguments

a) Set of names and corresponding reservation-success indicators.

6.6.2 Returned arguments

a) None.

6.6.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate had previously invoked the R*eserve Multiple Object Instance Names* service with those names.
d) Federate save not in progress.
e) Federate restore not in progress.

6.6.4 Postconditions

a) For each name, if its success indicator is true, the joined federate has been notified that it has obtained the unique right to register an object instance with that name.

6.6.5 Exceptions

a) Federate internal error.

6.6.6 Reference state charts

a) None.

6.7 *Release Multiple Object Instance Names* service

The *Release Multiple Object Instance Names* service indicates that the joined federate is attempting to relinquish the unique right to register object instances with the supplied names. Each supplied name shall be a name that is currently reserved by the joined federate, having previously been the subject of a successful invocation of the *Object Instance Name Reserved* or *Multiple Object Instance Name Reserved* service. Once an object instance name reservation is released, the name shall be available for any joined federate to reserve. Any unreserved name shall abort the release of all names in the list.

6.7.1 Supplied arguments

a) Name set.

6.7.2 Returned arguments

a) None.

6.7.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The supplied names are currently reserved by the joined federate, having previously been the subject of a successful invocation of a *Object Instance Name Reserved* or *Multiple Object Instance Name Reserved* service.
e) Save not in progress.
f) Restore not in progress.

6.7.4 Postconditions

a) The supplied names are no longer reserved by the joined federate.

6.7.5 Exceptions

a) An object instance name is not currently reserved.
b) Save in progress.
c) Restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

6.7.6 Reference state charts

a) None.

6.8 *Register Object Instance* service

The RTI shall create a federation execution-wide unique object instance handle and shall coadunate that handle with an instance of the supplied object class. All instance attributes of the object instance for which the corresponding class attributes are currently published by the registering joined federate shall be set as owned by the registering joined federate.

An object instance handle shall be uniform throughout the federation execution; i.e., with respect to a particular object instance, the handle provided to the joined federate that registers the object instance shall be the same handle provided to all joined federates that discover the object instance. The phrase “the same handle” shall denote handle equality rather than handle identity. Two handles shall be 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 shall also have equality between joined federates that are using different language APIs. The handles may be communicated between joined federates via instance attributes or interaction parameters.

If the optional object instance name argument is supplied, that name shall have been successfully reserved by the registering federate 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 a federation execution-wide unique name, and that name shall be coadunated with the object instance.

6.8.1 Supplied arguments

a) Object class designator.
b) Optional object instance name.

6.8.2 Returned arguments

a) Object instance handle.

6.8.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The object class is defined in the FDD.

e) The joined federate is publishing the object class.
f) If the optional object instance name argument is supplied, that name is reserved and not already coadunated with another object instance.
g) Federate save not in progress.
h) Federate restore not in progress.

6.8.4 Postconditions

a) The returned object instance handle is coadunated with the object instance.
b) The joined federate acquires ownership of the instance attributes that correspond to the currently published class attributes for the specified object class.
c) If the optional object instance name argument is supplied, that name is coadunated with the object instance.

6.8.5 Exceptions

a) The object instance name is already coadunated with another object instance (if optional object instance name argument is supplied).
b) The object instance name was not reserved (if optional object instance name argument is supplied).
c) The joined federate is not publishing the specified object class.
d) The object class is not defined in the FDD.
e) Federate save in progress.
f) Federate restore in progress.
g) The federate is not a federation execution member.
h) Not connected.
i) RTI internal error.

6.8.6 Reference state charts

a) Figure 10.
b) Figure 15.
c) Figure 16.

6.9 *Discover Object Instance †* service

The *Discover Object Instance †* service shall inform the joined federate to discover an object instance. Object instance discovery is described in 6.1. The object instance handle shall be unique to the federation execution and shall be uniform (see 6.8) throughout the federation execution.

If the Convey Producing Federate Switch for this joined federate is enabled, the producing joined federate argument shall contain the designator for the joined federate that registered the object instance. This producing joined federate may not own instance attributes that caused the discovery, and, in fact, it may be no longer joined to the federation execution.

6.9.1 Supplied arguments

a) Object instance handle.
b) Object class designator.
c) Object instance name.
d) Optional producing joined federate designator.

6.9.2 Returned arguments

a) None.

6.9.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) An object instance with the specified designator exists.
d) The joined federate is:
    1) Subscribed to some class attribute ***att*** at the specified object class, and the ***att***'s analogous instance attribute that is part of the specified object instance is either owned by another joined federate or owned by the RTI, and/or
    2) Subscribed with regions to some class attribute ***att*** at the specified object class, and the subscription region set overlaps the update region set of ***att***'s analogous instance attribute that is part of the specified object instance by the joined federate that owns that instance attribute.
e) The joined federate does not know about the object instance with the specified designator.
f) Federate save not in progress.
g) Federate restore not in progress.

6.9.4 Postconditions

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

6.9.5 Exceptions

a) Federate internal error.

6.9.6 Reference state charts

a) Figure 10.
b) Figure 15.
c) Figure 16.

6.10 *Update Attribute Values* service

The *Update Attribute Values* service shall provide instance attribute values to the federation. The joined federate shall be able to update only owned instance attributes. The joined federate should supply instance attribute values using the federation-agreed-upon update policies. This service, coupled with the *Reflect Attribute Values †* service, forms the primary data exchange mechanism supported by the RTI.

If preferred order type (see 8.1.1) of at least one instance attribute is TSO, the joined federate invoking this service is time-regulating, and the timestamp argument is present, this service shall return a federation-unique message retraction designator. Otherwise, no message retraction designator shall be returned. A timestamp value should be provided if the preferred order type of at least one instance attribute is TSO and the joined federate invoking this service is time-regulating.

The user-supplied tag argument shall be provided with all corresponding *Reflect Attribute Value †* service invocations, even multiple ones at the same reflecting joined federate.

If, at any wall-clock time between the invocation of this service and what would be the invocation of the corresponding *Reflect Attribute Values †* service for a given instance attribute that is supplied as an argument to this service, a potential joined federate is either

a) Unsubscribed to that instance attribute's corresponding class attribute at the known class of the object instance at the potential reflecting joined federate, or

b) Using a subscription region set for the instance attribute's corresponding class attribute at the known class of the object instance at the potential reflecting joined federate and this region set does not overlap the update region set of the instance attribute at the wall-clock time of update,

then the RTI shall not invoke the corresponding *Reflect Attribute Values †* service for that instance attribute at that joined federate.

If, at any wall-clock time between the invocation of this service and what would be the invocation of the corresponding *Reflect Attribute Values †* service for a given instance attribute that is supplied as an argument to this service, a potential reflecting joined federate is either

c) Subscribed to that instance attribute's corresponding class attribute at the known class of the object instance at the potential reflecting joined federate, or
d) Using a subscription region set for the instance attribute's corresponding class attribute at the known class of the object instance at the potential reflecting joined federate and this region set overlaps the update region set of the instance attribute at the wall-clock time of update,

then the RTI shall invoke the corresponding *Reflect Attribute Values †* service for the instance attribute at that joined federate, subject to the preconditions of the *Reflect Attribute Values †* service and the rules of time management.

6.10.1 Supplied arguments

a) Object instance designator.
b) Constrained set of attribute designator and value pairs.
c) User-supplied tag.
d) Optional timestamp.

6.10.2 Returned arguments

a) Optional message retraction designator.

6.10.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The joined federate owns the instance attributes for which values are provided.
e) The attributes are defined in the FDD.
f) An object instance with the specified designator exists.
g) The joined federate knows about the object instance with the specified designator.
h) The timestamp value shall be in accordance with the constraints stated in Clause 8 (if optional timestamp argument is supplied).
i) If the optional timestamp argument is supplied to this service invocation, the invoking joined federate is time-regulating, and it has invoked the *Delete Object Instance* service for the specified object instance with a timestamp argument (while time-regulating), then the update's timestamp shall be less than or equal to the delete's timestamp.
j) Federate save not in progress.
k) Federate restore not in progress.

6.10.4 Postconditions

a) The new instance attribute values have been supplied to the RTI.

6.10.5 Exceptions

a) The timestamp is invalid (if optional timestamp argument is supplied).
b) The joined federate does not own the specified instance attributes.
c) The specified class attributes are not available attributes of the instance object class.
d) The object instance is not known.
e) Federate save in progress.
f) Federate restore in progress.
g) The federate is not a federation execution member.
h) Not connected.
i) RTI internal error.

6.10.6 Reference state charts

a) Figure 15.

6.11 *Reflect Attribute Values †* service

The *Reflect Attribute Values †* service shall provide the joined federate with new values for the specified instance attributes. This service, coupled with the *Update Attribute Values* service, forms the primary data exchange mechanism supported by the RTI.

The user-supplied tag argument supplied to the corresponding *Update Attribute Values* service invocation shall be provided with all corresponding *Reflect Attribute Value †* service invocations, even multiple ones in the same reflecting joined federate.

The transportation type shall indicate the transportation type of the passel associated with this service invocation.

A timestamp shall be provided if one was specified in the corresponding *Update Attribute Values* service invocation. The receive message order type shall indicate the type (either RO or TSO) of the received message. The timestamp and receive message order type arguments shall be supplied together or not at all. The sent message order type shall indicate the type (either RO or TSO) of the sent message.

A retraction handle shall be provided if and only if the sent message order type is TSO.

If specified instance attributes have available dimensions and the Convey Region Designator Sets Switch for this joined federate is enabled, the Set of Sent Region Designators argument shall contain a copy of the update region realization (see 9.1.7), if any, that was used for update of the instance attributes at the joined federate, which invoked the corresponding *Update Attribute Values* service.

If the Convey Producing Federate Switch for this joined federate is enabled, the producing joined federate argument shall contain the designator for the joined federate that updated.

A joined federate invoking the *Update Attribute Values* service shall not receive the induced *Reflect Attribute Values †* service invocation, regardless of the subscription situation.

6.11.1 Supplied arguments

a) Object instance designator.
b) Constrained set of attribute designator and value pairs.
c) User-supplied tag.
d) Sent message order type.

e) Transportation type.
f) Optional timestamp.
g) Optional receive message order type.
h) Optional message retraction designator.
i) Optional set of sent region designators.
j) Optional producing joined federate designator.

6.11.2 Returned arguments

a) None.

6.11.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate knows about the object instance with the specified designator.
d) For each attribute designator specified, either
    1) This joined federate is subscribed to the class attribute at the known class of the object instance at this joined federate, or
    2) The joined federate has a subscription region set for the class attribute at the known class of the object instance at the subscribing joined federate and this region set overlaps the update region set of the instance attribute at the wall-clock time of update.
e) The joined federate did not send the corresponding update.
f) Federate save not in progress.
g) Federate restore not in progress.

6.11.4 Postconditions

a) The new instance attribute values have been supplied to the joined federate.

6.11.5 Exceptions

a) Federate internal error.

6.11.6 Reference state charts

a) Figure 15.

6.12 *Send Interaction* service

The *Send Interaction* service shall send an interaction into the federation. Only parameters that are available at the specified interaction class may be sent in a given interaction, as defined in the FDD. A federate is not required to send all available parameters of the interaction class with the interaction.

If preferred order type (see 8.1.1) of the interaction is TSO, the joined federate invoking this service is time-regulating, and the timestamp argument is present, this service shall return a federation-unique message retraction designator. Otherwise, no message retraction designator shall be returned. A timestamp value should be provided if the preferred order type of the interaction is TSO and the joined federate invoking this service is time-regulating.

If, at any wall-clock time between the invocation of this service for a particular interaction class argument and what would be the invocation of the corresponding *Receive Interaction †* service at a potential receiving

joined federate, the potential receiving joined federate is unsubscribed to the specified interaction class, the RTI shall not invoke the corresponding *Receive Interaction †* service at that joined federate.

If, at any wall-clock time between the invocation of this service for a particular interaction class and what would be the invocation of the corresponding *Receive Interaction †* service at a potential receiving joined federate, the potential receiving joined federate is either

a) Subscribed to the specified interaction class, or
b) Subscribed to the specified interaction class with regions,

then the RTI shall invoke the corresponding *Receive Interaction †* service at that joined federate, subject to the preconditions of the *Receive Interaction †* service and the rules of time management.

A joined federate invoking the *Send Interaction* service shall not receive the induced *Receive Interaction †* service invocation, regardless of the subscription/region situation.

The user-supplied tag argument shall be provided with all corresponding *Receive Interaction †* service invocations.

6.12.1 Supplied arguments

a) Interaction class designator.
b) Constrained set of interaction parameter designator and value pairs.
c) User-supplied tag.
d) Optional timestamp.

6.12.2 Returned arguments

a) Optional message retraction designator.

6.12.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The joined federate is publishing the interaction class.
e) The interaction class is defined in the FDD.
f) The parameters are defined in the FDD.
g) The timestamp value shall be in accordance with the constraints stated in Clause 8 (if optional timestamp argument is supplied).
h) Federate save not in progress (Certain MOM interactions can be sent during a save. See 11.4.2).
i) Federate restore not in progress (Certain MOM interactions can be sent during a restore. See 11.4.2).

6.12.4 Postconditions

a) The RTI has received the interaction.

6.12.5 Exceptions

a) The timestamp is invalid (if optional timestamp argument is supplied).
b) The joined federate is not publishing the specified interaction class.
c) The interaction parameter is not defined in FDD.
d) The interaction class is not defined in the FDD.
e) Federate save in progress.

f) Federate restore in progress.
g) The federate is not a federation execution member.
h) Not connected.
i) RTI internal error.

6.12.6 Reference state charts

a) Figure 13.

6.13 *Receive Interaction †* service

The *Receive Interaction †* service shall provide the joined federate with a sent interaction.

The user-supplied tag argument supplied to the corresponding *Send Interaction* or *Send Interaction With Regions* service invocation shall be provided with all corresponding *Receive Interaction †* service invocations.

The transportation type argument shall indicate the transportation type of the specified interaction, i.e., the transportation type of the published interaction at the joined federate that invoked the corresponding *Send Interaction* or *Send Interaction With Regions* service.

A timestamp shall be provided if one was specified in the corresponding *Send Interaction* or *Send Interaction With Regions* service invocation. The receive message order type shall indicate the type (either RO or TSO) of the received message. The timestamp and receive message order type arguments shall be supplied together or not at all. The sent message order type shall indicate the type (either RO or TSO) of the sent message.

A retraction handle shall be provided if and only if the sent message order type is TSO.

If the specified interaction has available dimensions and the Convey Region Designator Sets Switch for this joined federate is enabled, the Set of Sent Region Designators argument shall contain a copy of the update region realization (see 9.1.7), if any, that was supplied to the corresponding *Send Interaction With Regions* service invocation by the sending joined federate.

If the Convey Producing Federate Switch for this joined federate is enabled, the producing joined federate argument shall contain the designator for the sending joined federate

A joined federate invoking the *Send Interaction* or *Send Interaction With Regions* service shall not receive the induced *Receive Interaction †* service invocation, regardless of the subscription situation.

At most, one induced *Receive Interaction †* service invocation shall happen at a joined federate as a result of the invocation of the *Send Interaction* or *Send Interaction With Regions* service at another joined federate.

6.13.1 Supplied arguments

a) Interaction class designator.
b) Constrained set of interaction parameter designator and value pairs.
c) User-supplied tag.
d) Sent message order type.
e) Transportation type.
f) Optional timestamp.
g) Optional receive message order type.
h) Optional message retraction designator.

i) Optional set of sent region designators.
j) Optional producing joined federate designator.

6.13.2 Returned arguments

a) None.

6.13.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate
    1) Is subscribed to the interaction class, and/or
    2) Has a subscription region set for the interaction class and this region set overlaps the update region set that was used for sending the interaction.
d) The receiving joined federate did not send the corresponding interaction.
e) Federate save not in progress.
f) Federate restore not in progress.

6.13.4 Postconditions

a) The joined federate has received the interaction.

6.13.5 Exceptions

a) Federate internal error.

6.13.6 Reference state charts

a) Figure 13.

6.14 *Delete Object Instance* service

The *Delete Object Instance* service shall inform the federation that an object instance with the specified designator, which has the **HLAprivilegeToDeleteObject** instance attribute that is owned by the joined federate, is to be removed from the federation execution. Once the object instance is removed from the federation execution, the designator shall not be reused and all joined federates that owned attributes of the object instance shall no longer own those attributes. The object instance name may be reused for subsequently registered objects. The RTI shall use the *Remove Object Instance †* service to inform all other joined federates which know the object instance that the object instance has been deleted. The invoking joined federate shall own the **HLAprivilegeToDeleteObject** attribute of the specified object instance.

The preferred order type (see 8.1.1) of the sent message representing a *Delete Object Instance* service invocation shall be based on the preferred order type of the **HLAprivilegeToDeleteObject** attribute of the specified object instance. If preferred order type of the **HLAprivilegeToDeleteObject** attribute of the specified object instance is TSO, the joined federate invoking this service is time-regulating, and the timestamp argument is present, this service shall return a federation-unique message retraction designator. Otherwise, no message retraction designator shall be returned. A timestamp value should be provided if the preferred order type of the **HLAprivilegeToDeleteObject** attribute of the specified object instance is TSO and the joined federate invoking this service is time-regulating.

The user-supplied tag argument shall be provided with all corresponding *Remove Object Instance †* service invocations.

6.14.1 Supplied arguments

a) Object instance designator.
b) User-supplied tag.
c) Optional timestamp.

6.14.2 Returned arguments

a) Optional message retraction designator.

6.14.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.
e) The joined federate knows about the object instance with the specified designator.
f) The joined federate has the privilege to delete the object instance (it owns the **HLAprivilegeToDeleteObject** instance attribute).
g) The timestamp value shall be in accordance with the constraints stated in Clause 8 (if optional timestamp argument is supplied).
h) Federate save not in progress.
i) Federate restore not in progress.

6.14.4 Postconditions

a) The RTI has received the *Delete Object Instance* service invocation.
b) If the sent message order type of the *Delete Object Instance* service invocation is RO,
    1) The invoking joined federate may no longer update any previously owned attributes of the specified object instance and
    2) The object instance is no longer known at the invoking joined federate, no longer exists as far as the joined federate is concerned, and shall no longer be discovered at any other joined federate.
c) If the sent message order type of the *Delete Object Instance* service invocation is TSO,
    1) The invoking joined federate may no longer update any previously owned attributes of the specified object instance with sent message order type of TSO and a timestamp value greater than that specified in the *Delete Object Instance* service invocation
    2) Once the joined federate advances its logical time to or past the specified time, the object instance shall no longer be known at the invoking joined federate (and will no longer exist as far as the joined federate is concerned). If the joined federate disables time-regulation, the object instance immediately becomes unknown to the joined federate and ceases to exist as far as the joined federate is concerned. The object instance may still be discovered by other joined federates until their logical times are greater or equal to the specified time of the deletion.

6.14.5 Exceptions

a) The timestamp is invalid (if optional timestamp argument is supplied).
b) The joined federate does not own the delete privilege.
c) The object instance is not known.
d) Federate save in progress.
e) Federate restore in progress.
f) The federate is not a federation execution member.
g) Not connected.
h) RTI internal error.

6.14.6 Reference state charts

a) Figure 10.

6.15 *Remove Object Instance †* service

The *Remove Object Instance †* service shall inform the joined federate that an object instance has been deleted from the federation execution.

The user-supplied tag argument supplied to the corresponding *Delete Object Instance* service invocation shall be provided with all corresponding *Remove Object Instance †* service invocations.

A timestamp shall be provided if one was specified in the corresponding *Delete Object Instance* service invocation. The receive message order type shall indicate the type (either RO or TSO) of the received message. The timestamp and receive message order type arguments shall be supplied together or not at all. The sent message order type shall indicate the type (either RO or TSO) of the sent message.

A retraction handle shall be provided if the delete message had a sent order type of TSO. Otherwise, no retraction handle shall be provided.

If the Convey Producing Federate Switch for this joined federate is enabled, the producing joined federate argument shall contain the designator for the joined federate that registered the object instance. This producing joined federate may not own instance attributes that caused the discovery, and, in fact, it may even no longer be joined to the federation execution.

A joined federate invoking the *Delete Object Instance* service shall not receive the induced *Remove Object Instance †* service invocation.

6.15.1 Supplied arguments

a) Object instance designator.
b) User-supplied tag.
c) Sent message order type.
d) Optional timestamp.
e) Optional receive message order type.
f) Optional message retraction designator.
g) Optional producing joined federate designator.

6.15.2 Returned arguments

a) None.

6.15.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate knows about the object instance with the specified designator.
d) The receiving joined federate did not send the corresponding *Delete Object Instance* service.
e) Federate save not in progress.
f) Federate restore not in progress.

6.15.4 Postconditions

a) The joined federate has been notified to remove the object instance.
b) If one of the following is true:
   — A timestamp is not provided or
   — The receive message order type was RO or
   — The message is NOT the result of a *Flush Queue Request* service invocation,
   then the object instance is no longer known by the federate and no longer exists as far as the federate is concerned.
c) If a timestamp is provided, the receive message order type is TSO, and the message is the result of a *Flush Queue Request* service invocation, then once the joined federate advances it logical time to or past the specified time, the object instance shall no longer be known by the joined federate (and will no longer exist as far as the joined federate is concerned). If the joined federate successfully disables time-constrained then the object instance shall immediately become unknown by the joined federate (and will no longer exist as far as the joined federate is concerned).

6.15.5 Exceptions

a) Federate internal error.

6.15.6 Reference state charts

a) Figure 10.

6.16 *Local Delete Object Instance* service

The *Local Delete Object Instance* service shall inform the RTI that it shall treat the specified object instance as if the RTI had never notified the invoking joined federate to discover the object instance. The object instance shall not be removed from the federation execution. The joined federate shall not own any attributes of the specified object instance when invoking this service.

After the termination of invocation of this service, the specified object instance may subsequently be discovered by the invoking joined federate, at a possibly different class from previously known.

6.16.1 Supplied arguments

a) Object instance designator.

6.16.2 Returned arguments

a) None.

6.16.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.
e) The joined federate knows about the object instance with the specified designator.
f) The joined federate owns no attributes of the specified object instance.
g) Instance attributes of the specified object instance do not exist if the joined federate has either
   1) Invoked the *Attribute Ownership Acquisition* service, but has not yet received a corresponding invocation of the *Confirm Attribute Ownership Acquisition Cancellation †* service, the

Attribute Ownership Unavailable † service, or the *Attribute Ownership Acquisition Notification †* service, or

2) Invoked the *Attribute Ownership Acquisition If Available* service, but has not yet received an invocation of the *Attribute Ownership Unavailable †* service, received an invocation of the *Attribute Ownership Acquisition Notification †* service, or invoked the *Attribute Ownership Acquisition* service [after which condition 1) applies].

h) Federate save not in progress.
i) Federate restore not in progress.

6.16.4 Postconditions

a) The object instance does not exist with respect to the invoking joined federate.

6.16.5 Exceptions

a) Cannot local delete due to pending attempt to acquire instance attribute ownership.
b) The joined federate owns instance attributes.
c) The object instance is not known.
d) Federate save in progress.
e) Federate restore in progress.
f) The federate is not a federation execution member.
g) Not connected.
h) RTI internal error.

6.16.6 Reference state charts

a) Figure 10.

6.17 *Attributes In Scope †* service

The *Attributes In Scope †* service shall notify the joined federate that the specified attributes for the object instance are in scope for the joined federate. Generation of the *Attributes In Scope †* service advisory can be controlled using the *Enable/Disable Attribute Scope Advisory Switch* services. The *Attributes In Scope †* service shall be invoked only when the Attribute Scope Advisory Switch is enabled.

6.17.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.

6.17.2 Returned arguments

a) None.

6.17.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate knows about the object instance with the specified designator.
d) For each attribute designator specified, the joined federate either
   1) Is subscribed to the class attribute at the known class of the object instance at this joined federate, or

2) Has a subscription region set for the class attribute at the known class of the object instance at the subscribing joined federate.

e) The instance attributes have come into scope.
f) The Attribute Scope Advisory Switch is enabled.
g) Federate save not in progress.
h) Federate restore not in progress.

6.17.4 Postconditions

a) The joined federate has been advised that the instance attributes are in scope.

6.17.5 Exceptions

a) Federate internal error.

6.17.6 Reference state charts

a) None.

6.18 *Attributes Out Of Scope †* service

The *Attributes Out Of Scope †* service shall notify the joined federate that the specified attributes of the object instance are out of scope for the joined federate. Generation of the *Attributes Out Of Scope †* service advisory can be controlled using the *Enable/Disable Attribute Scope Advisory Switch* services. The *Attributes Out of Scope †* service shall be invoked only when the Attribute Scope Advisory Switch is enabled.

6.18.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.

6.18.2 Returned arguments

a) None.

6.18.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate knows about the object instance with the specified designator.
d) For each attribute designator specified, the joined federate either
   1) Is subscribed to the class attribute at the known class of the object instance at this joined federate, or
   2) Has a subscription region set for the class attribute at the known class of the object instance at the subscribing joined federate.
e) The instance attributes are no longer in scope.
f) The Attribute Scope Advisory Switch is enabled.
g) Federate save not in progress.
h) Federate restore not in progress.

6.18.4 Postconditions

a) The joined federate has been advised that the instance attributes are out of scope.

6.18.5 Exceptions

a) Federate internal error.

6.18.6 Reference state charts

a) None.

6.19 *Request Attribute Value Update* service

The *Request Attribute Value Update* service shall be used to stimulate the update of values of specified attributes. When this service is used, the RTI shall solicit the current values of the specified attributes from their owners using the *Provide Attribute Value Update †* service. The RTI shall not invoke the *Provide Attribute Value Update †* service for unowned instance attributes. When an object class is specified, the RTI shall solicit the values of the specified instance attributes for all object instances registered at that class and all subclasses of that class. When an object instance designator is specified, the RTI shall solicit the values of the specified instance attributes for the particular object instance. The timestamp of any resulting *Reflect Attribute Values †* service invocations is determined by the updating joined federate.

If the federate invoking this service also owns some of the instance attributes whose values are being requested, it shall not have the *Provide Attribute Value Update †* service invoked on it by the RTI. The request is taken as an implicit invocation of the provide service at the requesting federate for all qualified instance attributes that it owns.

The user-supplied tag argument shall be provided with all corresponding *Provide Attribute Value Update †* service invocations.

6.19.1 Supplied arguments

a) Object instance designator or object class designator.
b) Set of attribute designators.
c) User-supplied tag.

6.19.2 Returned arguments

a) None.

6.19.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified class attributes are available attributes of the known class of the specified object instance (when first argument is an object instance designator).
e) An object instance with the specified designator exists (when first argument is an object instance designator).
f) The joined federate knows about the object instance with the specified designator (when first argument is an object instance designator).
g) The specified object class is defined in the FDD (when first argument is an object class).

h) The specified class attributes are available attributes of the specified object class (when first argument is an object class).
i) Federate save not in progress.
j) Federate restore not in progress.

6.19.4 Postconditions

a) The request for the updated attribute values has been received by the RTI.

6.19.5 Exceptions

a) The class attribute is not available at the known class of the object instance.
b) The object class is not defined in FDD (if an object class designator was specified).
c) The object instance is not known (if an object instance designator was specified).
d) Federate save in progress.
e) Federate restore in progress.
f) The federate is not a federation execution member.
g) Not connected.
h) RTI internal error.

6.19.6 Reference state charts

a) None.

6.20 *Provide Attribute Value Update †* service

The *Provide Attribute Value Update †* service requests the current values for attributes owned by the joined federate for a given object instance. The owning joined federate responds to the *Provide Attribute Value Update †* service with an invocation of the *Update Attribute Values* service to provide the requested instance attribute values to the federation. The owning joined federate is not required to provide the values.

The user-supplied tag argument supplied to the corresponding *Request Attribute Value Update* service invocation shall be provided with all corresponding *Provide Attribute Value Update †* service invocations.

6.20.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.
c) User-supplied tag.

6.20.2 Returned arguments

a) None.

6.20.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate knows about the object instance with the specified designator.
d) The joined federate owns the specified instance attributes.
e) Federate save not in progress.
f) Federate restore not in progress.

6.20.4 Postconditions

a) The joined federate has been notified to provide updates of the specified instance attribute values.

6.20.5 Exceptions

a) Federate internal error.

6.20.6 Reference state charts

a) Figure 14.

6.21 *Turn Updates On For Object Instance †* service

The *Turn Updates On For Object Instance †* service shall indicate to the joined federate that the values of the specified attributes of the specified object instance are required somewhere in the federation execution. The update rate designator argument shall not be provided if there exists a subscription with the default, unspecified rate. If no such subscription exists, then the provided update rate designator argument shall be the update rate designator of the maximum subscribed rate.

The joined federate should commence with the federation-agreed-upon update scheme for the specified instance attributes. Generation of the *Turn Updates On For Object Instance †* service advisory can be controlled using the *Enable/Disable Attribute Relevance Advisory Switch* services. The *Turn Updates On For Object Instance †* service shall be invoked only when the Attribute Relevance Advisory Switch is enabled. The *Turn Updates On for Object Instance †* service shall also be invoked at owning joined federates whenever the maximum actively subscribed update rate changes.

6.21.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.
c) Optional maximum update rate designator.

6.21.2 Returned arguments

a) None.

6.21.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate owns the instance attributes.
d) The joined federate knows about the object instance with the specified designator.
e) For each attribute designator specified, there may be at least one joined federate in the federation execution for which the instance attribute is in scope via an active subscription.
f) The Attribute Relevance Advisory Switch is enabled.
g) Federate save not in progress.
h) Federate restore not in progress.

6.21.4 Postconditions

a) The joined federate has been notified by the RTI of the requirement for updates of the specified attributes of the specified object instance.

6.21.5 Exceptions

a) Federate internal error.

6.21.6 Reference state charts

a) Figure 15.

6.22 *Turn Updates Off For Object Instance †* service

The *Turn Updates Off For Object Instance †* service shall indicate to the joined federate that the values of the specified attributes of the object instance are not required anywhere in the federation execution. Generation of the *Turn Updates Off For Object Instance †* service advisory can be controlled using the *Enable/Disable Attribute Relevance Advisory Switch* services. The *Turn Updates Off For Object Instance †* service shall be invoked only when the Attribute Relevance Advisory Switch is enabled.

6.22.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.

6.22.2 Returned arguments

a) None.

6.22.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate owns the specified instance attributes.
d) The joined federate knows about the object instance with the specified designator.
e) For each attribute designator specified, there is no other joined federate in the federation execution for which the instance attribute is in scope via an active subscription.
f) The Attribute Relevance Advisory Switch is enabled.
g) Federate save not in progress.
h) Federate restore not in progress.

6.22.4 Postconditions

a) The joined federate has been notified by the RTI that updates of the specified attributes of the specified object instance are not required.

6.22.5 Exceptions

a) Federate internal error.

6.22.6 Reference state charts

a) Figure 14

6.23 *.Request Attribute Transportation Type Change* service

The transportation type for each attribute of an object instance shall be initialized from the object class description in the FDD. A joined federate may choose to change the transportation type during execution.

Invoking the *Request Attribute Transportation Type Change* service shall attempt to change the transportation type for all future *Update Attribute Values* service invocations for the specified attributes of the specified object instance only for the invoking joined federate. The RTI shall provide the achieved transportation type via the *Confirm Attribute Transportation Type Change †* service invocation. The change shall take effect with the RTI invocation of the responding *Confirm Attribute Transportation Type Change †* service. If the invoking joined federate loses ownership of an instance attribute after changing its transportation type and later acquires ownership of that instance attribute again, the transportation type shall be as defined in the FDD.

6.23.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.
c) Transportation type.

6.23.2 Returned arguments

a) None.

6.23.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.
e) The joined federate knows about the object instance with the specified designator.
f) The specified class attributes are available attributes of the known class of the specified object instance designator.
g) The specified instance attributes are not in the process of changing transportation type.
h) The joined federate owns the instance attributes.
i) The transportation type is valid (see 6.1.10).
j) Federate save not in progress.
k) Federate restore not in progress.

6.23.4 Postconditions

a) The RTI has been informed of the request to change the transportation type for the specified instance attributes.

6.23.5 Exceptions

a) One or more instance attributes are already in the process of changing transportation type.
b) The joined federate does not own the specified instance attributes.
c) The class attribute is not available at the known class of the object instance.
d) The object instance is not known.
e) Invalid transportation type.
f) Federate save in progress.
g) Federate restore in progress.
h) The federate is not a federation execution member.
i) Not connected.
j) RTI internal error.

6.23.6 Reference state charts

a) None.

6.24 *Confirm Attribute Transportation Type Change †* service

The *Confirm Attribute Transportation Type Change †* service shall be used to provide information on what transportation type was achieved for the specified object instance and attributes. This service shall be invoked by the RTI at a joined federate in response to a *Request Attribute Transportation Type Change* service invocation by the same joined federate.

6.24.1 Supplied arguments

a) Object instance designator.
b) Set of attributes designator.
c) Transportation type that was achieved.

6.24.2 Returned arguments

a) None.

6.24.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate has previously invoked the *Request Attribute Transportation Type Change* service for the object instance and attributes.
d) The transportation type is changed for the specified instance attributes.
e) Federate save not in progress.
f) Federate restore not in progress.

6.24.4 Postconditions

a) The joined federate has been informed about what transportation type has been achieved.

6.24.5 Exceptions

a) Federate internal error.

6.24.6 Reference state charts

a) None.

6.25 *Query Attribute Transportation Type* service

The *Query Attribute Transportation Type* service may be used by a joined federate to query the current transportation type of a single attribute of an object instance. The invoking joined federate may or may not own the instance attribute. The RTI shall provide the transportation type information via a *Report Attribute Transportation Type †* service invocation.

6.25.1 Supplied arguments

a) Object instance designator.

b) Attribute designator.

6.25.2 Returned arguments

a) None.

6.25.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.
e) The joined federate knows about the object instance with the specified designator.
f) The specified class attribute is an available attribute of the known class of the specified object instance designator.
g) Save not in progress.
h) Restore not in progress.

6.25.4 Postconditions

a) None.

6.25.5 Exceptions

b) The class attribute is not available at the known class of the object instance.
a) The object instance is not known.
b) Save in progress.
c) Restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

6.25.6 Reference state charts

a) None.

6.26 *Report Attribute Transportation Type †* service

The *Report Attribute Transportation Type †* service shall be used to provide the transportation type that is in use for the specified instance and instance attribute. If the instance attribute is unowned, the transportation type specified in the FDD shall be reported. This service shall be invoked by the RTI at a joined federate in response to a *Query Attribute Transportation Type* service invocation by the same joined federate.

6.26.1 Supplied arguments

a) Object instance designator.
b) Attribute designator.
c) Transportation type in use.

6.26.2 Returned arguments

a) None.

6.26.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate has previously invoked the *Query Attribute Transportation Type* service for the object instance and instance attribute.
d) Federate save not in progress.
e) Federate restore not in progress.

6.26.4 Postconditions

a) The joined federate has been informed about what transportation type is in use.

6.26.5 Exceptions

a) Federate internal error.

6.26.6 Reference state charts

a) None.

6.27 *Request Interaction Transportation Type Change* service

The transportation type for each interaction shall be initialized from the interaction class description in the FDD. A joined federate may choose to change the transportation type during execution. Invoking the *Request Interaction Transportation Type Change* service shall attempt to change the transportation type for all future *Send Interaction* and *Send Interaction With Regions* service invocations for the specified interaction class for the invoking joined federate only. The RTI shall provide the achieved transportation type via the *Confirm Interaction Transportation Type Change †* service invocation. The change shall take effect with the RTI invocation of the responding *Confirm Interaction Transportation Type Change †* service.

6.27.1 Supplied arguments

a) Interaction class designator.
b) Transportation type.

6.27.2 Returned arguments

a) None.

6.27.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The interaction class is defined in the FDD.
e) The specified interaction class is not in the process of changing transportation type.
f) The joined federate is publishing the interaction class.
g) The transportation type is valid (see 6.1.10).
h) Federate save not in progress.
i) Federate restore not in progress.

6.27.4 Postconditions

a) The RTI has been informed of the request to change the transportation type.

6.27.5 Exceptions

a) The interaction class is already in the process of changing transportation type.
b) The joined federate is not publishing the interaction class.
c) The interaction class is not defined in the FDD.
d) Invalid transportation type.
e) Federate save in progress.
f) Federate restore in progress.
g) The federate is not a federation execution member.
h) Not connected.
i) RTI internal error.

6.27.6 Reference state charts

a) None.

6.28 *Confirm Interaction Transportation Type Change †* service

The *Confirm Interaction Transportation Type Change †* service shall be used to provide information on what transportation type was achieved for the specified interaction class. This service shall be invoked by the RTI at a joined federate in response to an *Request Interaction Transportation Type Change* service invocation by the same joined federate.

6.28.1 Supplied arguments

a) Interaction class designator.
b) Transportation type that was achieved.

6.28.2 Returned arguments

a) None.

6.28.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate has previously invoked the *Request Interaction Transportation Type Change* service for the interaction class.
d) The transportation type is changed for the specified interaction class.
e) Federate save not in progress.
f) Federate restore not in progress.

6.28.4 Postconditions

a) The joined federate has been informed about what transportation type has been achieved.

6.28.5 Exceptions

a) Federate internal error.

6.28.6 Reference state charts

a) None.

6.29 *Query Interaction Transportation Type* service

The *Query Interaction Transportation Type* service may be used by a joined federate to query the current transportation type for an interaction for a specified joined federate. The RTI shall provide transportation type information via the *Report Interaction Transportation Type †* service invocation.

6.29.1 Supplied arguments

a) Federate designator.
b) Interaction class designator.

6.29.2 Returned arguments

a) None.

6.29.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The interaction class is defined in the FDD.
e) Save not in progress.
f) Restore not in progress.

6.29.4 Postconditions

a) None.

6.29.5 Exceptions

a) The interaction class is not defined in the FDD.
b) Save in progress.
c) Restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

6.29.6 Reference state charts

a) None.

6.30 *Report Interaction Transportation Type †* service

The *Report Interaction Transportation Type †* service shall be used to provide the transportation type that is used by the specified joined federate for the specified interaction class. If the specified joined federate does not publish the specified interaction, the transportation type specified in the FDD shall be reported. This service shall be invoked by the RTI at a joined federate in response to a *Query Interaction Transportation Type* service invocation by the same joined federate.

6.30.1 Supplied arguments

a) Federate designator.
b) Interaction class designator.
c) Transportation type in use.

6.30.2 Returned arguments

a) None.

6.30.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate has previously invoked the *Query Interaction Transportation Type* service for the interaction.
d) Federate save not in progress.
e) Federate restore not in progress.

6.30.4 Postconditions

a) The joined federate has been informed about what transportation type is in use.

6.30.5 Exceptions

a) Federate internal error,

6.30.6 Reference state charts

a) None.

7. Ownership management

7.1 Overview

Ownership management is used by joined federates and the RTI to transfer ownership of instance attributes among joined federates. The ability to transfer ownership of instance attributes among joined federates shall be required to support the cooperative modeling of a given object instance across a federation. Only the joined federate that owns an instance attribute shall

— Invoke the *Update Attribute Values* service to provide a new value for that instance attribute
— Receive invocations of the *Provide Attribute Value Update †* service for that instance attribute
— Receive invocations of the *Turn Updates On For Object Instance †* and *Turn Updates Off For Object Instance †* services pertaining to that instance attribute

Figure 16 depicts the ways that ownership of a single instance attribute may be established from the viewpoint of a given joined federate. This diagram is complete insofar as all transitions shown represent legal operations, and transitions that are not shown represent illegal operations. The RTI shall generate exceptions if illegal operations are invoked.

An instance attribute shall not be owned by more than one joined federate, and an instance attribute may be unowned by all joined federates. The RTI shall be responsible for attempting to find an owner for instance attributes that are left unowned as a result of a joined federate resignation or unconditional 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. The RTI shall invoke the *Request Attribute Ownership Assumption †* service once at each joined federate that is both eligible to own the instance attribute and eligible to receive a *Request Attribute Ownership Assumption †* service invocation for that instance attribute until the instance attribute becomes owned. From a given joined federate's perspective, every instance attribute shall be either owned or unowned. Hence, within the state machine depicted in Figure 16, the owned and unowned states are exclusive.

Upon registration of an object instance, the registering joined federate shall own all instance attributes of that object instance for which the joined federate is publishing the corresponding class attributes at the registered class of the object instance. All other instance attributes of that object instance shall be unowned by all joined federates. Upon discovery of an object instance, the discovering joined federate shall not own any instance attributes of that object instance. If a joined federate does not own an instance attribute, it shall not own that instance attribute until it has received an *Attribute Ownership Acquisition Notification †* service invocation for it.

Ownership of an instance attribute shall be transferred from one joined federate to another either by the owning joined federate requesting to divest itself of the instance attribute or by a non-owning joined federate requesting to acquire it. Regardless of whether an instance attribute changes ownership as a result of being divested by its owner or acquired by a non-owner, the instance attribute shall change ownership only as a result of explicit service invocations by the owning and acquiring joined federates. The RTI shall not take ownership away from, nor give it to, a joined federate without the joined federate's consent, except through the use of the following MOM interactions:

— HLAmanager.HLAfederate.HLAadjust.HLAmodifyAttributeState
— HLAmanager.HLAfederate.HLAservice.HLAunpublishObjectClassAttributes
— HLAmanager.HLAfederate.HLAservice.HLAunconditionalAttributeOwnershipDivestiture
— HLAmanager.HLAfederate.HLAservice.HLAresignFederationExecution

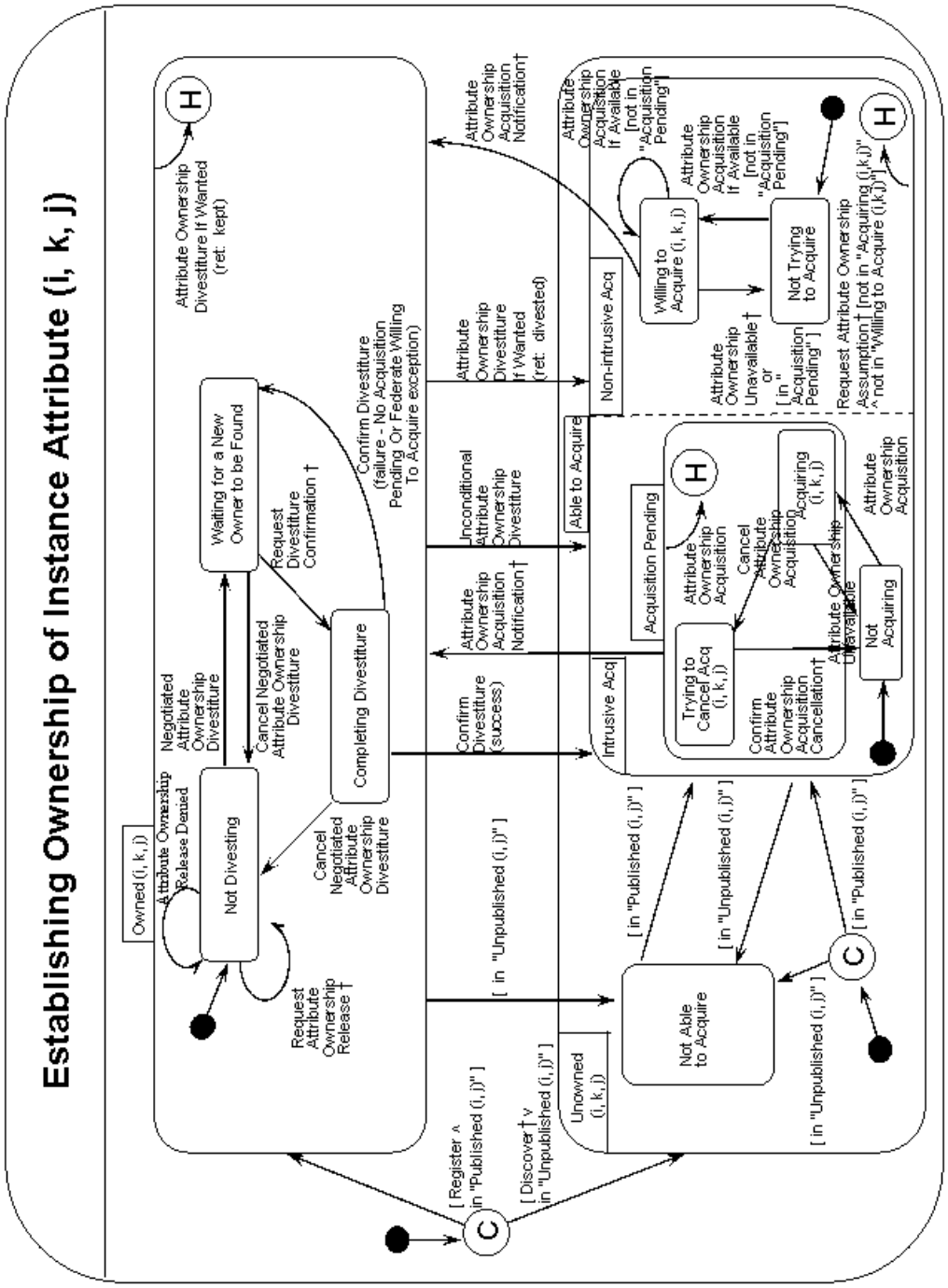

Figure 16—Establishing ownership of instance attribute (*i*, *k*, *j*)

7.1.1 Ownership and publication

The ownership of an instance attribute is closely related to whether that instance attribute's corresponding class attribute is published at the known class of the instance attribute. In other words, the ownership state machine in Figure 16, which operates in parallel with the publication state machine in Figure 11, also shares interdependencies with the publication state machine. A joined federate shall be publishing a class attribute at the known class of an object instance in order to own the corresponding instance attribute of that object instance. Therefore, the following are true:

a) A joined federate shall be publishing a class attribute at the known class of an object instance before it may become the owner of the corresponding instance attribute of that object instance. This interdependency between ownership and publication is expressed in Figure 16 by the Not Able to Acquire state, the [in "Unpublished (*i*, *j*)"] and [in "Published (*i*, *j*)"] transitions in the Unowned (*i*, *k*, *j*) state, and the conditional transition into the Owned (*i*, *k*, *j*) and Unowned (*i*, *k*, *j*) states from the start state.
b) If the joined federate that owns an instance attribute stops publishing the corresponding class attribute at the known class of the instance attribute, the instance attribute shall immediately become unowned. This interdependency between ownership and publication is expressed in Figure 16 by the transition from the Owned (*i*, *k*, *j*) state to the Unowned (*i*, *k*, *j*) state that is labeled [in "Unpublished (*i*, *j*)"]. As depicted by the guard on the transition from the Published (*i*, *j*) state to the Unpublished (*i*, *j*) state in the publication state machine shown in Figure 11, a joined federate shall not stop publication of a class attribute at a given class if there is an object instance that has that class as its known class and that has a corresponding instance attribute that is in either the Acquisition Pending or Willing to Acquire (*i*, *k*, *j*) state at that joined federate. In other words, a joined federate shall not stop publishing a class attribute at a given class if there is an object instance that has that class as its known class and that has a corresponding instance attribute for which the joined federate has invoked the following:
    1) *Attribute Ownership Acquisition* service, but has not yet received an invocation of the *Confirm Attribute Ownership Acquisition Cancellation †* service, the *Attribute Ownership Unavailable †* service, or the *Attribute Ownership Acquisition Notification †* service
    2) *Attribute Ownership Acquisition If Available* service, but has not yet received an invocation of the *Attribute Ownership Unavailable †* service, received an invocation of the *Attribute Ownership Acquisition Notification †* service, or invoked the *Attribute Ownership Acquisition service* [after which condition 1) (above) applies].

7.1.2 Ownership transfer

An instance attribute that is successfully divested shall become unowned by the divesting joined federate. If an instance attribute is unowned, its corresponding class attribute at the known class of the instance attribute may be either published or unpublished. If the class attribute is published at that class, the joined federate shall be eligible to acquire the corresponding instance attribute, and it may be offered ownership of that instance attribute by the RTI via the *Request Attribute Ownership Assumption †* service. There are five ways in which an owning joined federate may attempt to divest itself of an instance attribute and two ways in which a non-owning joined federate may attempt to acquire one.

7.1.2.1 Divestiture

The five actions that a joined federate may take to cause an instance attribute that it owns to become unowned are as follows:

a) The joined federate may invoke the *Unconditional Attribute Ownership Divestiture* service, in which case, the instance attribute shall immediately become unowned by that joined federate and, in fact, by all joined federates.

b) The joined federate may invoke the *Negotiated Attribute Ownership Divestiture* service, which notifies the RTI that the joined federate wishes to divest itself of the instance attribute, providing that the RTI can locate a joined federate that is willing to own the instance attribute. If any joined federates are in the process of trying to acquire the instance attribute, these joined federates are willing to own the instance attribute. The RTI can try to identify other joined federates that are willing to own the instance attribute by invoking the *Request Attribute Ownership Assumption †* service at all joined federates that are not in the process of trying to acquire the instance attribute, but that are publishing the instance attribute's corresponding class attribute at the publishing joined federate's known class of the instance attribute. If the RTI is able to locate a joined federate that is willing to acquire the instance attribute, the RTI shall notify the divesting joined federate by invoking the *Request Divestiture Confirmation †* service. The divesting joined federate may then complete the negotiated divestiture using the *Confirm Divestiture* service, which causes ownership of the instance attribute to be lost.

c) The joined federate may invoke the *Attribute Ownership Divestiture If Wanted* service, which notifies the RTI that the joined federate wishes to divest itself of ownership of the instance attribute, providing that another joined federate is attempting to acquire ownership of it. This service invocation has a return argument that the RTI shall use to indicate the set of instance attributes for which the joined federate has divested ownership. If an instance attribute is not in the returned instance attribute set, it was not divested, and the request to divest this instance attribute shall be concluded. If the *Attribute Ownership Divestiture If Wanted* service returns with the designated instance attribute among the set of released instance attributes, the instance attribute shall be unowned by the invoking joined federate. [In Figure 16, the transition from the owned to the unowned state via an *Attribute Ownership Divestiture If Wanted* service invocation is labeled "*Attribute Ownership Divestiture If Wanted* (ret: divested)." This is a convenience notation indicating that the instance attribute in question is a member of the returned instance attribute set.]

d) The joined federate may stop publishing the instance attribute's corresponding class attribute at the known class of the instance attribute; this step shall result in the instance attribute immediately becoming unowned by that joined federate and, in fact, by all joined federates.

e) The joined federate may resign from the federation execution. When a joined federate successfully resigns from the federation execution with the Unconditionally Divest Attributes option, all of the instance attributes that are owned by that joined federate shall become unowned by that joined federate and, in fact, by all joined federates. This transition is not depicted in Figure 16 because it occurs at a joined federate, rather than at an instance attribute, level of operation.

Of the five ways a joined federate may divest itself of an instance attribute, only the *Negotiated Attribute Ownership Divestiture* service may be canceled. A *Negotiated Attribute Ownership Divestiture* service invocation shall remain pending until either the instance attribute becomes unowned or the divesting joined federate cancels the divestiture request by invoking the *Cancel Negotiated Attribute Ownership Divestiture* service. Cancellation of the divestiture shall be guaranteed to be successful.

Of the five ways a joined federate may divest itself of an instance attribute, three ways (invocation of the *Unconditional Attribute Ownership Divestiture* service, a request to stop publication of the instance attribute's corresponding class attribute at the known class of the instance attribute, and invocation of the *Resign Federation Execution* service) shall result in the instance attribute becoming unowned by all joined federates. When either the *Negotiated Attribute Ownership Divestiture* or the *Attribute Ownership Divestiture If Wanted* service is used, the RTI shall guarantee that immediately after the owning joined federate loses ownership of the instance attribute, another joined federate shall be granted ownership of it. For purposes of determining an instance attribute's scope, the instance attribute shall be considered to be continuously owned during its transfer of ownership from the divesting joined federate to the acquiring joined federate via either the *Negotiated Attribute Ownership Divestiture* or the *Attribute Ownership Divestiture If Wanted* service.

When the value of an instance attribute is being updated by a joined federate using TSO messages and that joined federate is initiating a negotiated divestiture of that instance attribute, the divesting and receiving joined federates should be aware of potential causality or reflection anomalies.

Because the divestiture is not associated with a timestamp, the logical time of the receiving joined federate at which ownership of the instance attribute is acquired may be earlier or later than that of the divesting joined federate, giving either a gap or an overlap in the time intervals on the HLA time axis for which the joined federates own the instance attribute.

Some issues to consider are as follows:

a) There may be points on the HLA time axis at which neither joined federate owns the instance attribute. This situation may result in an interval of timestamps for which no updates to the value of the instance attribute are provided, unless special steps are taken by one or both of the joined federates.
b) There may be points on the HLA time axis at which both joined federates own the instance attribute. This situation may result in updates to the value of the instance attribute being produced by both joined federates that contain the same timestamp, unless special steps are taken by one or both of the joined federates.
c) There may be TSO updates sent by the divesting joined federate that have not been delivered to other joined federates (i.e., they are queued in the RTI) when the divestiture occurs. Some of these outstanding TSO updates may be retractable by the divesting joined federate; others may not be retractable. See restrictions in 8.21 for details.
d) There may be additional information that the divesting joined federate needs to transfer along with ownership of the instance attribute.

7.1.2.2 Acquisition

The RTI shall provide both of the following two ways for a joined federate that is publishing a class attribute at a given class to acquire an analogous instance attribute of an object instance that has that class as the known class at the joined federate attempting to acquire the attribute:

a) The joined federate may invoke the *Attribute Ownership Acquisition* service, which shall inform the RTI that it shall invoke the *Request Attribute Ownership Release †* service at the joined federate that owns the designated instance attribute.
b) The joined federate may invoke the *Attribute Ownership Acquisition If Available* service, which shall inform the RTI that it wants to acquire the designated instance attribute only if it is already unowned by all joined federates or if it is in the process of being divested by its owner.

The first method of acquisition can be thought of as an intrusive acquisition because the RTI will notify the joined federate that owns the instance attribute that another joined federate wants to acquire it and request that the owning joined federate release the instance attribute for acquisition by the requesting joined federate. The second method of acquisition can be thought of as a nonintrusive acquisition because the RTI will not notify the owning joined federate of the request to acquire the instance attribute. The *Attribute Ownership Acquisition* service can also be thought of as taking precedence over the *Attribute Ownership Acquisition If Available* service. A joined federate that has invoked the *Attribute Ownership Acquisition* service and is in the Acquisition Pending state shall not invoke the *Attribute Ownership Acquisition If Available* service. If a joined federate that has invoked the *Attribute Ownership Acquisition If Available* service and is in the Willing to Acquire state invokes the *Attribute Ownership Acquisition* service, that joined federate shall enter the Acquisition Pending state.

An *Attribute Ownership Acquisition* service invocation may be explicitly canceled, but an *Attribute Ownership Acquisition If Available* service invocation may not be explicitly canceled. When a joined federate invokes the *Attribute Ownership Acquisition If Available* service, either the *Attribute Ownership*

Acquisition Notification † service or the *Attribute Ownership Unavailable †* service shall be invoked at that joined federate in response. If the instance attribute is unowned by all joined federates (and no other joined federate is attempting to acquire it), the *Attribute Ownership Acquisition Notification †* service shall be invoked. If the joined federate is in the Not Divesting state with respect to the instance attribute, the *Attribute Ownership Unavailable †* service shall be invoked. Otherwise, either the *Attribute Ownership Acquisition Notification †* service or the *Attribute Ownership Unavailable †* service may be invoked, depending on what service, if any, the owning joined federate invokes.

When a joined federate invokes the *Attribute Ownership Acquisition* service invocation, on the other hand, this request shall remain pending until the instance attribute is acquired (as indicated by an invocation of the *Attribute Ownership Acquisition Notification †* service), acquisition of the instance attribute is denied (as indicated by an invocation of the *Attribute Ownership Unavailable †* service), or the joined federate successfully cancels the acquisition request. A joined federate may attempt to cancel the acquisition request by invoking the *Cancel Attribute Ownership Acquisition* service. The *Cancel Attribute Ownership Acquisition* service is not guaranteed to be successful. If it is successful, the RTI shall indicate this success to the canceling joined federate by invoking the *Confirm Attribute Ownership Acquisition Cancellation †* service. If it fails, the RTI shall indicate this failure to the canceling joined federate by invoking the *Attribute Ownership Acquisition Notification †* service, thereby granting ownership of the instance attribute to the joined federate.

An *Attribute Ownership Acquisition* service invocation shall override an *Attribute Ownership Acquisition If Available* service invocation. In other words, a joined federate that has invoked the *Attribute Ownership Acquisition If Available* service may, before it receives an invocation of either the *Attribute Ownership Acquisition Notification †* service or the *Attribute Ownership Unavailable †* service, invoke the *Attribute Ownership Acquisition* service. In this case, the *Attribute Ownership Acquisition If Available* service request shall be implicitly canceled, and the *Attribute Ownership Acquisition* service request shall remain pending until the instance attribute is acquired, acquisition of the instance attribute is denied, or the joined federate successfully cancels the acquisition request. A joined federate that has invoked the *Attribute Ownership Acquisition* service, but has not yet received an invocation of the *Attribute Ownership Acquisition Notification †* service, the *Attribute Ownership Unavailable †* service, or the *Confirm Attribute Ownership Acquisition Cancellation †* service, shall not invoke the *Attribute Ownership Acquisition If Available* service.

7.1.3 Privilege to delete object

All object classes shall have an available attribute called **HLAprivilegeToDeleteObject**. As with all other available attributes, a joined federate shall be publishing the **HLAprivilegeToDeleteObject** class attribute at the known class of an object instance in order to be eligible for ownership of a corresponding **HLAprivilegeToDeleteObject** instance attribute.

A **HLAprivilegeToDeleteObject** instance attribute shall be transferable among joined federates. All ownership management services for **HLAprivilegeToDeleteObject** instance attributes shall act the same as they are for all other instance attributes. The reason that a joined federate may want to own the **HLAprivilegeToDeleteObject** instance attribute, however, is different. Ownership of a typical instance attribute shall give a joined federate the privilege to provide new values for that instance attribute. Ownership of the **HLAprivilegeToDeleteObject** instance attribute of an object instance shall give the joined federate the additional right to delete that object instance from the federation execution.

7.1.4 User-supplied tags

Several of the ownership management services take a user-supplied tag as an argument. These arguments shall be provided as a mechanism for conducting information between joined federates that could be used to implement priority or other schemes. Although the content and use of these tags are outside of the scope of this specification, the RTI shall pass these user-supplied tags from joined federates that are trying to acquire

an instance attribute to the joined federate that owns the instance attribute and from the joined federate that is trying to divest itself of an instance attribute to the joined federates that are able to acquire the instance attribute. In particular, the following apply:

— The user-supplied tag that is present in the *Negotiated Attribute Ownership Divestiture* service shall be present in any resulting *Request Attribute Ownership Assumption †* service invocations.
— The user-supplied tag that is present in the *Confirm Divestiture* service shall be present in any resulting *Attribute Ownership Acquisition Notification †* service invocations.
— The user-supplied tag that is present in the *Attribute Ownership Acquisition* service shall be present in any resulting *Request Attribute Ownership Release †* service invocations.

If an RTI-invoked service is not the result of a joined 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 shall be an empty (zero-length) array.

The preceding requirement applies not only to RTI-invoked services related to ownership management, but to all RTI-invoked services, such as MOM HLAmanager.HLAfederate.HLAreport interactions. All received interactions include mandatory user-supplied tag arguments; therefore, the user-supplied tag arguments in MOM HLAmanager.HLAfederate.HLAreport interactions shall be present, but empty.

7.1.5 Sets of attribute designators

Although many of the ownership management services take a set of instance attributes as an argument, the RTI treats ownership management operations on a per-instance-attribute basis. The fact that some ownership management service invocations take sets of instance attributes as an argument is a feature provided to joined federate designers for convenience. A single request with an instance attribute set as an argument can result in multiple responses pertaining to disjoint subsets of those instance attributes. For example, a single *Negotiated Attribute Ownership Divestiture* service invocation that has a set of instance attributes as an argument could result in multiple *Request Divestiture Confirmation †* service invocations. If one instance attribute in the set of instance attributes provided as an argument to an ownership management service invocation violates the preconditions of the service, an exception shall be generated, and the entire service invocation shall fail. The RTI shall treat all joined federate requests to divest or acquire attributes separately. Responses to or requests of joined federates shall not contain information from separate joined federate requests.

7.2 *Unconditional Attribute Ownership Divestiture* service

The *Unconditional Attribute Ownership Divestiture* service shall notify the RTI that the joined federate no longer wants to own the specified instance attributes of the specified object instance. This service shall immediately relieve the divesting joined federate of the ownership and cause the instance attribute(s) to go (possibly temporarily) into the unowned state, without regard to the existence of an accepting joined federate.

For each instance attribute that becomes unowned as a result of invocation of this service, if no joined federates are in either the Acquiring state or the 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 state or the 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 other joined federates (joined federates other than the divesting joined federate) that are both eligible to own the instance attribute and not in either the Acquiring state or the Willing to Acquire state with respect to the specified instance attribute. As long as the instance attribute remains unowned, the RTI shall try to

identify joined federates (other than the divesting joined federate) that are willing to own the instance attribute; but once the instance attribute becomes owned, the RTI shall not invoke the *Request Attribute Ownership Assumption †* service at any additional joined federates.

7.2.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.

7.2.2 Returned arguments

a) None.

7.2.3 Preconditions.

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.
e) The joined federate knows about the object instance with the specified designator.
f) The joined federate owns the specified instance attributes.
g) Federate save not in progress.
h) Federate restore not in progress.

7.2.4 Postconditions

a) The joined federate no longer owns the specified instance attributes.

7.2.5 Exceptions

a) The joined federate does not own the instance attribute.
b) The class attribute is not available at the known class of the object instance.
c) The object instance is not known.
d) Federate save in progress.
e) Federate restore in progress.
f) The federate is not a federation execution member.
g) Not connected.
h) RTI internal error.

7.2.6 Reference state charts

a) Figure 16.

7.3 *Negotiated Attribute Ownership Divestiture* service

The *Negotiated Attribute Ownership Divestiture* service shall notify the RTI that the joined federate no longer wants to own the specified instance attributes of the specified object instance. Ownership shall be transferred only if some joined federate(s) accept. When the RTI finds joined federates willing to accept ownership of any or all of the instance attributes, it shall inform the divesting joined federate using the *Request Divestiture Confirmation †* service (supplying the appropriate instance attributes as arguments). The divesting joined federate may then complete the negotiated divestiture by invoking the *Confirm Divestiture* service to inform the RTI of which instance attributes it is divesting ownership. The invoking joined federate shall continue its update responsibility for the specified instance attributes until it divests ownership. The

joined federate may receive one or more *Request Divestiture Confirmation †* service invocations for each invocation of this service because different joined federates may wish to become the owner of different instance attributes.

A request to divest ownership shall remain pending until the request is completed (via the *Request Divestiture Confirmation †* and *Confirm Divestiture* services), the requesting joined federate successfully cancels the request (via the *Cancel Negotiated Attribute Ownership Divestiture* service), or the joined federate divests itself of ownership by other means (e.g., the *Attribute Ownership Divestiture If Wanted* or *Unpublish Object Class Attributes* service). A second negotiated divestiture for an instance attribute already in the process of a negotiated divestiture shall result in an RTI-generated exception.

If no joined federates are in either the Acquiring state or the 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 state or the 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 joined federates that are both eligible to own the instance attribute and not in either the Acquiring state or the Willing to Acquire state with respect to the specified instance attribute.

7.3.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.
c) User-supplied tag.

7.3.2 Returned arguments

a) None.

7.3.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.
e) The joined federate knows about the object instance with the specified designator.
f) The joined federate owns the specified instance attributes.
g) The specified instance attributes are not in the negotiated divestiture process.
h) Federate save not in progress.
i) Federate restore not in progress.

7.3.4 Postconditions

a) No change has occurred in instance attribute ownership.
b) The RTI has been notified of the joined federate’s request to divest ownership of the specified instance attributes.

7.3.5 Exceptions

a) The instance attribute is already in the negotiated divestiture process.
b) The joined federate does not own the instance attribute.
c) The class attribute is not available at the known class of the object instance.
d) The object instance is not known.

e) Federate save in progress.
f) Federate restore in progress.
g) The federate is not a federation execution member.
h) Not connected.
i) RTI internal error.

7.3.6 Reference state charts

a) Figure 16.

7.4 *Request Attribute Ownership Assumption †* service

The *Request Attribute Ownership Assumption †* service shall inform the joined federate that the specified instance attributes are available for transfer of ownership to the joined federate. The RTI shall supply an object instance designator and set of attribute designators. The joined federate may return a subset of the supplied attribute designators for which it is willing to assume ownership via the *Attribute Ownership Acquisition* service or the *Attribute Ownership Acquisition If Available* service. In the case that the supplied instance attributes are unowned as a result of a joined federate invoking the *Unconditional Attribute Ownership Divestiture* service, the divesting joined federate shall not be asked to assume ownership.

7.4.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.
c) User-supplied tag.

7.4.2 Returned arguments

a) None.

7.4.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate knows about the object instance with the specified designator.
d) The joined federate is publishing the corresponding class attributes at the known class of the specified object instance.
e) The joined federate does not own the specified instance attributes.
f) The joined federate is not in either the Acquiring state or the Willing to Acquire state for this instance attribute.
g) Federate save not in progress.
h) Federate restore not in progress.

7.4.4 Postconditions

a) Instance attribute ownership has not changed.
b) The joined federate has been informed of the set of instance attributes for which the RTI is requesting that the joined federate assume ownership.

7.4.5 Exceptions

a) Federate internal error.

7.4.6 Reference state charts

a) Figure 16.

7.5 *Request Divestiture Confirmation †* service

The *Request Divestiture Confirmation †* service shall notify the joined federate that new owners have been found for the specified instance attributes and that the negotiated divestiture of the specified instance attributes can now be completed at the joined federate's discretion. The joined federate can complete the negotiated divestiture of the specified instance attributes using the *Confirm Divestiture* service, divest ownership of the instance attributes by some other means (e.g., using the *Unconditional Attribute Ownership Divestiture* service), or cancel the negotiated divestiture using the *Cancel Negotiated Attribute Ownership Divestiture* service. Ownership of the specified instance attributes is not lost upon invocation of this service; ownership shall be divested only if the joined federate confirms its intent to divest the instance attributes or divests ownership by some other means.

7.5.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.

7.5.2 Returned arguments

a) None.

7.5.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate knows about the object instance with the specified designator.
d) The joined federate owns the specified instance attributes.
e) The joined federate has previously attempted to divest ownership of the specified instance attributes and has not subsequently canceled that request.
f) Federate save not in progress.
g) Federate restore not in progress.

7.5.4 Postconditions

a) The joined federate has been informed that new owners have been found for the specified instance attributes and that ownership of the instance attributes shall be divested as soon as the joined federate confirms its intent to divest.

7.5.5 Exceptions

a) Federate internal error.

7.5.6 Reference state charts

a) Figure 16.

7.6 *Confirm Divestiture* service

The *Confirm Divestiture* service shall inform the RTI that the joined federate wants to complete negotiated divestiture for the specified instance attributes. After invocation of this service, the joined federate shall not own any of the specified instance attributes, and the RTI shall give ownership of the instance attributes to another joined federate.

If a joined 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 joined 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.

7.6.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.
c) User-supplied tag.

7.6.2 Returned arguments

a) None.

7.6.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.
e) The joined federate knows about the object instance with the specified designator.
f) The joined federate owns the specified instance attributes.
g) The joined federate has been asked to confirm its request for a negotiated divestiture of the specified instance attributes.
h) There is at least one joined federate in the federation that is in either the Acquisition Pending state or the Willing to Acquire state with respect to the specified instance attribute.
i) Federate save not in progress.
j) Federate restore not in progress.

7.6.4 Postconditions

a) The joined federate no longer owns the specified instance attributes.

7.6.5 Exceptions

a) There is no joined federate that has an acquisition pending or that is willing to acquire the instance attribute.
b) The joined federate had not previously been asked to confirm its request for negotiated divestiture of the instance attributes.
c) The joined federate does not own the instance attributes.
d) The class attribute is not available at the known class of the object instance.
e) The object instance is not known.
f) Federate save in progress.
g) Federate restore in progress.
h) The federate is not a federation execution member.

i) Not connected.
j) RTI internal error.

7.6.6 Reference state charts

a) Figure 16.

7.7 *Attribute Ownership Acquisition Notification †* service

The *Attribute Ownership Acquisition Notification †* service shall notify the joined federate that it now owns the specified set of instance attributes. The joined federate may then begin updating those instance attribute values. The joined federate may receive multiple notifications for a single invocation of the *Attribute Ownership Acquisition* service because the joined federate may wish to become the owner of instance attributes owned by different joined federates.

7.7.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.
c) User-supplied tag.

7.7.2 Returned arguments

a) None.

7.7.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate knows about the object instance with the specified designator.
d) The joined federate is publishing the corresponding class attributes at the known class of the specified object instance.
e) The joined federate has previously attempted to acquire ownership of the specified instance attributes.
f) Prior to the joined federate acquiring ownership, the specified instance attributes were not owned by any joined federate in the federation execution or were in the process of being divested by their owner.
g) Federate save not in progress.
h) Federate restore not in progress.

7.7.4 Postconditions

a) The joined federate has been notified that it now owns the specified instance attributes.
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 invoked any of the following:
    1) *Attribute Ownership Acquisition* service and has not yet received a corresponding invocation of the *Confirm Attribute Ownership Acquisition Cancellation †* service, the *Attribute Ownership Unavailable †* service, or the *Attribute Ownership Acquisition Notification †* service, or
    2) *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) *Attribute Ownership Acquisition If Available* service and has subsequently invoked the *Attribute Ownership Acquisition* service [after which condition 1) applies].

c) The affected instance attributes may have an implicit out-of-scope situation at the joined federate.

7.7.5 Exceptions

a) Federate internal error.

7.7.6 Reference state charts

a) Figure 16.

7.8 *Attribute Ownership Acquisition* service

The *Attribute Ownership Acquisition* service shall request the ownership of the specified instance attributes of the specified object instance. If a specified instance attribute is owned by another joined federate and that owning joined 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 joined 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. The joined federate may receive one or more *Attribute Ownership Acquisition Notification †* invocations for each invocation of this service.

A request to acquire ownership shall remain pending until the request is granted (via the *Attribute Ownership Acquisition Notification †* service), the acquisition is denied (via the *Attribute Ownership Unavailable †* service), or the requesting joined federate successfully cancels the request (via the *Cancel Attribute Ownership Acquisition* and *Confirm Attribute Ownership Acquisition Cancellation †* services).

If a joined 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 joined federate invokes the *Attribute Ownership Acquisition* service for an instance attribute for which the joined federate is in the Acquiring state, the joined federate shall continue to be in the Acquiring state with respect to that instance attribute, and the joined federate that owns the instance attribute shall not receive a corresponding *Request Attribute Ownership Release †* service invocation. If there are additional instance attributes in the attribute set that is an argument to the *Attribute Ownership Acquisition* service, the joined federate shall enter the Acquiring state with respect to those instance attributes, assuming that they meet all of the preconditions of the *Attribute Ownership Acquisition* service and that the joined federate is not already in the Acquiring state or the Trying to Cancel Acquisition state with respect to those instance attributes. The joined federate that owns those instance attributes shall receive a corresponding *Request Attribute Ownership Release †* service invocation for those attributes, if appropriate.

Likewise, if a joined federate invokes the *Attribute Ownership Acquisition* service for an instance attribute for which the joined federate is in the Trying to Cancel Acquisition state, the joined federate shall continue to be in the Trying to Cancel Acquisition state with respect to that instance attribute, and the joined federate that owns the instance attribute shall not receive a corresponding *Request Attribute Ownership Release †* service invocation. If there are additional instance attributes in the attribute set that is an argument to the *Attribute Ownership Acquisition* service, the joined federate shall enter the Acquiring state with respect to those instance attributes, assuming that they meet all of the preconditions of the *Attribute Ownership Acquisition* service and that the joined federate is not already in the Acquiring state or the Trying to Cancel Acquisition state with respect to those instance attributes. The joined federate that owns those instance

attributes shall receive a corresponding *Request Attribute Ownership Release †* service invocation for those attributes, if appropriate.

7.8.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.
c) User-supplied tag.

7.8.2 Returned arguments

a) None.

7.8.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.
e) The joined federate knows about the object instance with the specified designator.
f) The joined federate is publishing the corresponding class attributes at the known class of the specified object instance.
g) The joined federate does not own the specified instance attributes.
h) Federate save not in progress.
i) Federate restore not in progress.

7.8.4 Postconditions

a) The RTI has been informed of the joined federate's request to acquire ownership of the specified instance attributes.
b) The joined federate shall not stop publishing the corresponding class attributes at the known class of the specified object instance until it has received a corresponding invocation of the *Confirm Attribute Ownership Acquisition Cancellation †* service, the *Attribute Ownership Unavailable †* service, or the *Attribute Ownership Acquisition Notification †* service.

7.8.5 Exceptions

a) The joined federate is not publishing the class attribute at the known class of the object instance.
b) The joined federate is not publishing the object class.
c) The joined federate already owns the instance attribute.
d) The class attribute is not available at the known class of the object instance.
e) The object instance is not known.
f) Federate save in progress.
g) Federate restore in progress.
h) The federate is not a federation execution member.
i) Not connected.
j) RTI internal error.

7.8.6 Reference state charts

a) Figure 16.

7.9 *Attribute Ownership Acquisition If Available* service

The *Attribute Ownership Acquisition If Available* service shall request the ownership of the specified instance attributes of the specified object instance only if the instance attribute is unowned by all joined federates or it is in the process of being divested by its owner. If a specified instance attribute is owned by another joined federate, the RTI shall not invoke the *Request Attribute Ownership Release †* service for that instance attribute at the owning joined federate. 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.

If a joined 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 joined federate invokes the *Attribute Ownership Acquisition If Available* service for an instance attribute for which the joined federate 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, the joined federate shall enter the Willing to Acquire state with respect to those instance attributes, assuming that they meet all of the preconditions of the *Attribute Ownership Acquisition If Available* service and that the joined federate is not already in the Willing to Acquire state.

7.9.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.

7.9.2 Returned arguments

a) None.

7.9.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.
e) The joined federate knows about the object instance with the specified designator.
f) The joined federate is publishing the corresponding class attributes at the known class of the specified object instance.
g) The joined federate does not own the specified instance attributes.
h) For each of the specified instance attributes, it is not the case that the joined federate has invoked the *Attribute Ownership Acquisition* service, but has not yet received an invocation of the *Confirm Attribute Ownership Acquisition Cancellation †* service, the *Attribute Ownership Unavailable †* service, or the *Attribute Ownership Acquisition Notification †* service.
i) Federate save not in progress.
j) Federate restore not in progress.

7.9.4 Postconditions

a) The RTI has been informed of the joined federate's request to acquire ownership of the specified instance attributes.
b) The joined federate shall not stop publishing the corresponding class attributes at the known class of the specified object instance until it has received a corresponding invocation of either the *Attribute Ownership Unavailable †* service or the *Attribute Ownership Acquisition Notification †* service.

7.9.5 Exceptions

a) The joined federate is already acquiring the instance attribute.
b) The joined federate is not publishing the class attribute at the known class of the object instance.
c) The joined federate is not publishing the object class.
d) The joined federate already owns the instance attribute.
e) The class attribute is not available at the known class of the object instance.
f) The object instance is not known.
g) Federate save in progress.
h) Federate restore in progress.
i) The federate is not a federation execution member.
j) Not connected.
k) RTI internal error.

7.9.6 Reference state charts

a) Figure 16.

7.10 *Attribute Ownership Unavailable †* service

The *Attribute Ownership Unavailable †* service shall inform the joined federate that the specified instance attributes were not available for ownership acquisition.

7.10.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.

7.10.2 Returned arguments

a) None.

7.10.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate knows about the object instance with the specified designator.
d) The joined federate had requested ownership acquisition for the specified instance attributes.
e) The joined federate does not own the specified instance attributes.
f) Federate save not in progress.
g) Federate restore not in progress.

7.10.4 Postconditions

a) The joined federate has been informed that the specified instance attributes were not available for ownership acquisition.
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
    1) Invoked the *Attribute Ownership Acquisition* service and has not yet received a corresponding invocation of the *Confirm Attribute Ownership Acquisition Cancellation †* service, the *Attribute Ownership Unavailable †* 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].

7.10.5 Exceptions

a) Federate internal error.

7.10.6 Reference state charts

a) Figure 16.

7.11 *Request Attribute Ownership Release †* service

The *Request Attribute Ownership Release †* service shall request that the joined federate release ownership of the specified instance attributes of the specified object instance. The *Request Attribute Ownership Release †* service shall provide an object instance designator and set of attribute designators and shall be invoked only as the result of an *Attribute Ownership Acquisition* service invocation by some other joined federate. The joined federate may return the subset of the supplied instance attributes for which it is willing to release ownership via the *Attribute Ownership Divestiture If Wanted* service, the *Unconditional Attribute Ownership Divestiture* service, or the *Negotiated Attribute Ownership Divestiture* service. The joined federate may terminate the acquisition without releasing ownership by informing the RTI that it is unwilling to divest ownership via the *Attribute Ownership Release Denied* service.

7.11.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.
c) User-supplied tag.

7.11.2 Returned arguments

a) None.

7.11.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate knows about the object instance with the specified designator.
d) The joined federate owns the specified instance attributes.
e) Federate save not in progress.
f) Federate restore not in progress.

7.11.4 Postconditions

a) The joined federate has been informed of the set of instance attributes for which the RTI is requesting the joined federate to release ownership.

7.11.5 Exceptions

a) Federate internal error.

7.11.6 Reference state charts

a) Figure 16.

7.12 *Attribute Ownership Release Denied* service

The *Attribute Ownership Release Denied* service shall notify the RTI that the joined federate is unwilling to divest itself of ownership of the specified instance attributes if another joined federate is attempting to acquire ownership of them. The joined federate may use this service to provide an answer to the question posed as a result of the invocation of *Request Attribute Ownership Release †*, or it may use this service for any other reason. Invocation of this service for attributes that have no pending acquisition has no effect. Invocation of the *Attribute Ownership Release Denied* service shall cause the RTI to terminate any pending acquisitions and invoke *Attribute Ownership Unavailable* at the acquiring joined federates.

7.12.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators for which the joined federate is unwilling to divest ownership.

7.12.2 Returned arguments

a) None.

7.12.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.
e) The joined federate knows about the object instance with the specified designator.
f) The joined federate owns the specified instance attributes.
g) Federate save not in progress.
h) Federate restore not in progress.

7.12.4 Postconditions

a) No change has occurred in instance attribute ownership.
b) The RTI has been informed of the set of instance attributes that the joined federate is not willing to divest.

7.12.5 Exceptions

a) The joined federate does not own the instance attribute.
b) The class attribute is not available at the known class of the object instance.
c) The object instance is not known.
d) Federate save in progress.
e) Federate restore in progress.
f) The federate is not a federation execution member.
g) Not connected.
h) RTI internal error.

7.12.6 Reference state charts

a) Figure 15.

7.13 *Attribute Ownership Divestiture If Wanted* service

The *Attribute Ownership Divestiture If Wanted* service shall notify the RTI that the joined federate is willing to divest itself of ownership of the specified instance attributes if another joined federate is attempting to acquire ownership of them. The joined federate may use this service to provide an answer to the question posed as a result of the invocation of *Request Attribute Ownership Release †*, or it may use this service for any other reason. The returned argument shall indicate the instance attributes for which ownership was actually divested, and it shall be viewed as the conclusion of this attempted divestiture for all supplied instance attributes that are not in the returned argument. Any instance attributes in the returned argument shall be a subset of the instance attributes in the supplied argument.

7.13.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators for which the joined federate is willing to divest ownership.

7.13.2 Returned arguments

a) Set of attribute designators for which ownership has actually been divested.

7.13.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.
e) The joined federate knows about the object instance with the specified designator.
f) The joined federate owns the specified instance attributes.
g) Federate save not in progress.
h) Federate restore not in progress.

7.13.4 Postconditions

a) Ownership is divested for the instance attributes in the returned argument set.

7.13.5 Exceptions

a) The joined federate does not own the instance attribute.
b) The class attribute is not available at the known class of the object instance.
c) The object instance is not known.
d) Federate save in progress.
e) Federate restore in progress.
f) The federate is not a federation execution member.
g) Not connected.
h) RTI internal error.

7.13.6 Reference state charts

a) Figure 16.

7.14 *Cancel Negotiated Attribute Ownership Divestiture* service

The *Cancel Negotiated Attribute Ownership Divestiture* service shall notify the RTI that the joined federate no longer wants to divest ownership of the specified instance attributes.

7.14.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.

7.14.2 Returned arguments

a) None.

7.14.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.
e) The joined federate knows about the object instance with the specified designator.
f) The joined federate owns the specified instance attributes.
g) The specified instance attributes were candidates for divestiture.
h) Federate save not in progress.
i) Federate restore not in progress.

7.14.4 Postconditions

a) The specified instance attributes are unavailable for divestiture.

7.14.5 Exceptions

a) The instance attribute was not a candidate for divestiture.
b) The joined federate does not own the instance attribute.
c) The class attribute is not available at the known class of the object instance.
d) The object instance is not known.
e) Federate save in progress.
f) Federate restore in progress.
g) The federate is not a federation execution member.
h) Not connected.
i) RTI internal error.

7.14.6 Reference state charts

a) Figure 16.

7.15 *Cancel Attribute Ownership Acquisition* service

The *Cancel Attribute Ownership Acquisition* service shall notify the RTI that the joined federate no longer wants to acquire ownership of the specified instance attributes. This service shall always receive one of three replies from the RTI. The first form of reply, *Confirm Attribute Ownership Acquisition Cancellation*, shall indicate that the request to acquire ownership of the specified instance attributes has been successfully canceled. The second form of reply, *Attribute Ownership Acquisition Notification †*, shall indicate that the

request to acquire ownership of the specified instance attributes was not canceled before the joined federate acquired ownership of the instance attributes. The third form of reply, *Attribute Ownership Unavailable †*, shall indicate that the request to acquire ownership of the specified instance attributes was not canceled before the joined federate was denied ownership of the instance attributes. The joined federate may receive multiple forms of reply in response to a single *Cancel Attribute Ownership Acquisition* service invocation since the cancellation may succeed for some of the supplied instance attributes and fail for others. This service shall be used only to cancel requests to acquire ownership of instance attributes that were made via the *Attribute Ownership Acquisition* service. Requests made via the *Attribute Ownership Acquisition If Available* service shall not be explicitly canceled. They may, however, be overridden by an invocation of the *Attribute Ownership Acquisition* service.

7.15.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.

7.15.2 Returned arguments

a) None.

7.15.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.
e) The joined federate knows about the object instance with the specified designator.
f) The joined federate does not own the specified instance attributes.
g) The joined federate is attempting to acquire ownership of the specified instance attributes.
h) Federate save not in progress.
i) Federate restore not in progress.

7.15.4 Postconditions

a) The RTI has been notified that joined federate no longer wants to acquire ownership of the specified instance attributes.

7.15.5 Exceptions

a) The joined federate was not attempting to acquire ownership of the instance attribute.
b) The joined federate already owns the instance attribute.
c) The class attribute is not available at the known class of the object instance.
d) The object instance is not known.
e) Federate save in progress.
f) Federate restore in progress.
g) The federate is not a federation execution member.
h) Not connected.
i) RTI internal error.

7.15.6 Reference state charts

a) Figure 16.

7.16 *Confirm Attribute Ownership Acquisition Cancellation †* service

The *Confirm Attribute Ownership Acquisition Cancellation †* service shall inform the joined federate that the specified instance attributes are no longer candidates for ownership acquisition.

7.16.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.

7.16.2 Returned arguments

a) None.

7.16.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate knows about the object instance with the specified designator.
d) The joined federate had attempted to cancel an ownership acquisition request for the specified instance attributes.
e) The joined federate does not own the specified instance attributes.
f) Federate save not in progress.
g) Federate restore not in progress.

7.16.4 Postconditions

a) The federate has been notified that the specified instance attributes are no longer candidates for acquisition by the joined federate.
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
    1) Invoked the *Attribute Ownership Acquisition* service and has not yet received a corresponding invocation of the *Confirm Attribute Ownership Acquisition Cancellation †* service, the *Attribute Ownership Unavailable †* 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].

7.16.5 Exceptions

a) Federate internal error.

7.16.6 Reference state charts

a) Figure 16.

7.17 *Query Attribute Ownership* service

The *Query Attribute Ownership* service shall be used to determine the owner of the specified instance attribute. The RTI shall provide the instance attribute owner information via the *Inform Attribute Ownership †* service invocation.

7.17.1 Supplied arguments

a) Object instance designator.
b) Attribute designator.

7.17.2 Returned arguments

a) None.

7.17.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.
e) The joined federate knows about the object instance with the specified designator.
f) The corresponding class attribute is an available attribute of the known class of the specified object instance.
g) Federate save not in progress.
h) Federate restore not in progress.

7.17.4 Postconditions

a) The request for instance attribute ownership information has been received by the RTI.

7.17.5 Exceptions

a) The class attribute is not available at the known class of the object instance.
b) The object instance is not known.
c) Federate save in progress.
d) Federate restore in progress.
e) The federate is not a federation execution member.
f) Not connected.
g) RTI internal error.

7.17.6 Reference state charts

a) None.

7.18 *Inform Attribute Ownership †* service

The *Inform Attribute Ownership †* service shall be used to provide ownership information for the specified instance attribute. This service shall be invoked by the RTI in response to a *Query Attribute Ownership* service invocation by a joined federate. This service shall provide the joined federate with a designator of the instance attribute owner (if the instance attribute is owned) or an indication that the instance attribute is available for acquisition.

7.18.1 Supplied arguments

a) Object instance designator.
b) Attribute designator.
c) Ownership designator (i.e., joined federate, RTI, or unowned).

7.18.2 Returned arguments

a) None.

7.18.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate knows about the object instance with the specified designator.
d) The corresponding class attribute is an available attribute of the known class of the specified object instance.
e) The joined federate has previously invoked the *Query Attribute Ownership* service and has not yet received an *Inform Attribute Ownership †* service invocation in response.
f) Federate save not in progress.
g) Federate restore not in progress.

7.18.4 Postconditions

a) The joined federate has been informed of the instance attribute ownership.

7.18.5 Exceptions

a) Federate internal error.

7.18.6 Reference state charts

a) None.

7.19 *Is Attribute Owned By Federate* service

The *Is Attribute Owned By Federate* service shall be used to determine if the specified instance attribute of the specified object instance designator is owned by the invoking joined federate. The service shall return a boolean value indicating ownership status of the specified instance attribute.

7.19.1 Supplied arguments

a) Object instance designator.
b) Attribute designator.

7.19.2 Returned arguments

a) Instance attribute ownership indicator.

7.19.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.

c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.
e) The joined federate knows about the object instance with the specified designator.
f) The corresponding class attribute is an available attribute of the known class of the specified object instance.
g) Federate save not in progress.
h) Federate restore not in progress.

7.19.4 Postconditions

a) The joined federate has the requested ownership information.

7.19.5 Exceptions

a) The class attribute is not available at the known class of the object instance.
b) The object instance is not known.
c) Federate save in progress.
d) Federate restore in progress.
e) The federate is not a federation execution member.
f) Not connected.
g) RTI internal error.

7.19.6 Reference state charts

a) None.

8. Time management

8.1 Overview

The time management services and associated mechanisms provide a federation execution with the means to order the delivery of messages throughout the federation execution. Use of these mechanisms permits messages sent by different joined federates to be delivered in a consistent order to any joined federate in the federation execution that is to receive those messages.

Time in the system being modeled shall be represented in the federation as points along the HLA time axis. A joined federate can associate both itself and some of its activities with points on the HLA time axis. A joined federate's association with the HLA time axis is referred to as the logical time of the joined federate. An association of a joined federate's activities with the HLA time axis is denoted by assigning timestamps to the messages representing those activities. Timestamps and logical times are both represented by the same datatype and, therefore, can be compared, but they are referenced using different terms in order to help clarify whether a given federation time value applies to a joined federate or to a joined federate's activities.

Each joined federate may advance along the HLA time axis during the course of the execution. Such joined federate logical time advances may be constrained by the progress of other joined federates, or they may be unconstrained.

Time management is concerned with the mechanisms for controlling the advancement of each joined federate along the HLA time axis. In general, logical time advances shall be coordinated with object management services so that information is delivered to joined federates in a causally correct and ordered fashion.

A joined federate that becomes time-regulating may associate some of its activities (e.g., updating instance attribute values and sending interactions) with points on the HLA time axis. It shall do so by assigning timestamps to the messages representing its activities; these timestamps correspond to the points on the HLA time axis with which the activities are associated. A joined federate that is time-constrained is interested in receiving the messages representing these activities (e.g., reflecting instance attribute values and receiving interactions) in a federation-wide timestamped order. Use of the time management services allows this type of coordination among time-regulating and time-constrained joined federates in an execution. The coordination is achieved by various constraints on joined federate activities described in this clause.

The activities of joined federates that are neither time-regulating nor time-constrained (the default state of all joined federates upon joining a federation execution) shall not be coordinated with other joined federates by the RTI, and such joined federates need not make use of any of the time management services. Such joined federates may, however, optionally place timestamps on the messages representing their activities (e.g., updating instance attribute values and sending interactions) to indicate when these actions occur during the federation execution. In this case, the RTI simply passes the specified timestamp, unchanged, to joined federate(s) receiving the message.

The representation of logical times and timestamps shall be specified to the RTI via the time representation abstract datatype (TRADT) (see IEEE Std 1516.2-2010, clause 4.7) and should be documented in the time representation table of the federation execution's FDD. Federation executions requiring multiple time representations within a single execution (e.g., logical time and wall-clock time values) shall realize this functionality explicitly within their TRADT definition, e.g., by using a "time type" field to differentiate between different types of time values.

8.1.1 Messages

The manner in which HLA services are coordinated with the HLA time axis shall be through the concept of messages as follows:

a) Invocation of the *Update Attribute Values* service, *Send Interaction* service, *Send Interaction with Regions* service, or *Delete Object Instance* service by a joined federate shall be called sending a message. (Invocation of the *Update Attribute Values* service may actually result in the sending of multiple messages as each passel resulting from the invocation shall be considered a separate message.)
b) Invocation of the *Reflect Attribute Values †* service, *Receive Interaction †* service, or *Remove Object Instance †* service at a joined federate shall be called receiving a message.

Messages sent by one joined federate typically result in one or more other joined federates receiving a corresponding message. The mapping from one sent message to one or more received messages shall follow the descriptions for *Update Attribute Values* service (see 6.10), *Reflect Attribute Values †* service (see 6.11), *Send Interaction* service (see 6.12), *Received Interaction †* service (see 6.13), *Delete Object Instance* service (see 6.14), *Remove Object Instance †* service (see 6.15), and *Send Interaction With Regions* service (see 9.12). For example, a sent message representing an *Update Attribute Values* service invocation shall result only in received messages representing *Reflect Attribute Values †* service invocations at the appropriate joined federates depending on the normal ownership/subscription properties.

8.1.2 Message order type

Each message, sent or received, shall be either a timestamp order (TSO) message or an receive order (RO) message (see 8.1.11). The order type of a message shall be determined by several factors:

a) Preferred order type: The preferred order type of a message shall be the same as the preferred order type of the data contained in the message (instance attribute values or interactions). Each class attribute and interaction class shall be provided with a preferred order type in the FDD that indicates the order type (TSO or RO) that shall be used when sending messages carrying values for instances of these classes. In the case of sent messages representing a *Delete Object Instance* service invocation, the preferred order type of the message shall be based on the preferred order type of the **HLAprivilegeToDeleteObject** attribute of the specified object instance. Joined federates may use the *Change Attribute Order Type* service to change the preferred order type of instance attributes; the preferred order type of class attributes may not be changed during a federation execution. Joined federates may use the *Change Interaction Order Type* service to change the preferred order type of interaction classes.
b) Presence of a timestamp: Each of the services that corresponds to sending a message shall include a timestamp argument if the preferred order type of any instance attribute or interaction class specified in the message is TSO and the joined federate is time-regulating. This timestamp argument is always optional, but it is recommended in such cases as the occurrence of a preferred order type of TSO and a time-regulating joined federate indicate an intent for TSO messages to be sent.
c) Joined federate's time status: Whether a joined federate is time-regulating shall determine whether a joined federate can send TSO messages. Similarly, whether a joined federate is time-constrained shall determine whether the joined federate can receive TSO messages.
d) Sent message order type: The order type of a received message shall depend on the order type of the corresponding sent message.

These factors shall be considered together when determining if a given message is sent or received as a TSO message or as an RO message.

The order type of a sent message shall be determined by the preferred order type of the message at the sending joined federate, regardless of whether that joined federate is time-regulating and whether a

timestamp was supplied in the service invocation that sends the message. Table 1 illustrates how the order type of a sent message shall be determined.

Table 1—Order type of a sent message

Preferred order type?	Is the sending joined federate time-regulating?	Was a timestamp supplied?	Order type of sent message	Message retraction designator returned
RO	No	No	RO	No
RO	No	Yes	RO	No
RO	Yes	No	RO	No
RO	Yes	Yes	RO	No
TSO	No	No	RO	No
TSO	No	Yes	RO	No
TSO	Yes	No	RO	No
TSO	Yes	Yes	TSO	Yes

Table 2 illustrates how the order type of a received message shall be determined. The order type of a received message shall be determined by whether that joined federate is time-constrained and by the order type of the corresponding sent message. As shown in these two tables, a message retraction designator is provided to both the sender and the receiver if the sent message order type is TSO, regardless of the order type of the received message.

Table 2—Order type of a received message

Is the receiving joined federate time-constrained?	Order type of corresponding sent message?	Order type of received message?	Message retraction designator provided
No	RO	RO	No
No	TSO	RO	Yes
Yes	RO	RO	No
Yes	TSO	TSO	Yes

Because of the above rule defining the order type of a received message, the RTI shall sometimes convert a sent TSO message to a received RO message at some receiving joined federates. The need for such conversions shall be considered on a per-joined-federate basis. Thus, the received messages at different joined federates that correspond to the same sent message may be of different order types. Sent RO messages shall never be converted to received TSO messages.

Except when the *Flush Queue Request* service is used, messages that are received as TSO messages shall be received only by a given joined federate in TSO, regardless of the joined federates from which the messages originate and regardless of the sequence in which the messages were sent. Thus, two TSO messages with different timestamps shall always be received by each joined federate in the same order. Multiple TSO messages having the same timestamp may be received in an indeterminate order.

Messages that are received as RO messages shall be received in an arbitrary order. A timestamp shall be provided with the received message if one was specified when the message was sent.

8.1.3 Logical time

Each joined federate, upon joining an execution, shall be assigned a logical time. A joined federate's logical time shall initially be set to the initial time on the HLA time axis (the initial federation time of the TRADT). The logical time of a joined federate shall only advance; thus, a joined federate may request to advance only to a logical time that is greater than or equal to its current logical time. In order for a joined federate to advance its logical time, it shall request an advance explicitly. A joined federate shall request to advance its logical time only by invoking one of the following services:

— *Time Advance Request*
— *Time Advance Request Available*
— *Next Message Request*
— *Next Message Request Available*
— *Flush Queue Request*

The advance shall not occur until the RTI issues a grant via the *Time Advance Grant †* service. In general, at any instant during an execution, different joined federates may be at different logical times.

Joined federates also may become time-regulating and/or time-constrained. The logical times of joined federates that are time-regulating shall be used to constrain the advancement of the logical times of joined federates that are time-constrained.

8.1.4 Time-regulating joined federates

Only time-regulating joined federates may send TSO messages. A joined federate shall request to become time-regulating by invoking the *Enable Time Regulation* service. The RTI shall subsequently make the joined federate time-regulating by invoking the *Time Regulation Enabled †* service at that joined federate. A joined federate shall cease to be time-regulating whenever it invokes the *Disable Time Regulation* service.

Each time-regulating joined federate shall provide a lookahead when becoming time-regulating. Lookahead shall be a nonnegative value that establishes the lowest value of timestamps that can be sent in TSO messages by the joined federate. Specifically, a time-regulating joined federate shall not send a TSO message that contains a timestamp less than its current logical time plus its current lookahead. Once established, changes to a joined federate's lookahead may be requested only by using the *Modify Lookahead* service.

A time-regulating joined federate with a lookahead of zero shall be subject to an additional restriction. If such a joined federate has advanced its logical time by use of *Time Advance Request* or *Next Message Request* service, it shall not send TSO messages that contain timestamps less than or equal to its logical time (plus its lookahead, which is zero), rather than the usual less-than restriction. Subsequent use of a time advancement service that moves the joined federate's logical time forward lifts this additional restriction (and imposes the restriction of the subsequent advancement service). For example, if a zero lookahead joined federate were to invoke *Time Advance Request* service (t_1) and to follow this step with an invocation of *Time Advance Request Available* service (t_1), that joined federate would still have the additional restriction. After the *Time Advance Request Available* service is granted, it still may not send any TSO messages with a timestamp less than or equal to t_1 (the *Time Advance Request* restriction) because the second advance did not advance the joined federate's logical time.

A time-regulating joined federate need not send TSO messages in TSO, but all TSO messages that it sends shall be received by other joined federates in timestamped order (if they are received as TSO messages) except when the *Flush Queue Request* service is used.

8.1.5 Time-constrained joined federates

Only time-constrained joined federates can receive messages as TSO messages. A joined federate shall request to become time-constrained by invoking the *Enable Time Constrained* service. The RTI shall subsequently make the joined federate time-constrained by invoking the *Time Constrained Enabled †* service at that joined federate. A joined federate shall cease to be time-constrained whenever it invokes the *Disable Time Constrained* service.

In order to ensure that a time-constrained joined federate shall never receive a TSO message with a timestamp less than its logical time, a bound is placed on each time-constrained joined federate that limits how far it can advance its logical time. The bound ensures that a time-constrained joined federate cannot advance its logical time past a point at which TSO messages could still be sent by another joined federate. If a time-constrained joined federate request to advance its logical time beyond its bound, the time advance shall not be granted until its bound has increased beyond the logical time to be granted.

This bound on advancement is expressed in terms of a value called the Greatest Available Logical Time (GALT). Each joined federate shall have a GALT that expresses the greatest logical time to which the RTI guarantees it can grant an advance without having to wait for other joined federates to advance. In the case of some time advance services (*Time Advance Request* and *Next Message Request*) the joined federate shall be granted only to logical times that are less than its GALT. In the case of other time advance services (*Time Advance Request Available*, *Next Message Request Available*, and *Flush Queue Request*), the joined federate shall be granted only to logical times that are less than or equal to its GALT.

A joined federate's GALT is calculated by the RTI and is based on factors such as the logical time, lookahead, and requests to advance the logical time of time-regulating joined federates. If there are no other time-regulating joined federates in the execution, however, the joined federate's GALT is undefined, indicating that the bound is such that the joined federate may advance to any point without having to wait for another joined federate to advance.

While GALT is used only by the RTI to bound the advancement of time-constrained joined federates, nonconstrained joined federates also have a GALT. For a nonconstrained joined federate, GALT expresses the bound that would apply to that joined federate if it were to become time-constrained.

Each joined federate shall have another value that builds on the idea of GALT called Least Incoming Timestamp (LITS). A joined federate's LITS shall express the smallest timestamp that the joined federate could (but not necessarily will) receive in the future in a TSO message. A joined federate's LITS is calculated by the RTI and is based on the joined federate's GALT and any queued TSO messages that may later be received by the joined federate. If the joined federate's GALT is undefined and there are no queued TSO messages that the joined federate could receive, the joined federate's LITS shall also be undefined.

LITS is more useful for time-constrained joined federates, but it applies to all joined federates[7]. LITS is useful for joined federates wishing to know the timestamp of the next TSO message that they may have to process.

8.1.6 Advancing logical time

A joined federate is permitted to advance its logical time only by requesting a time advancement from the RTI (as indicated in 8.1.3). Its logical time shall not actually be advanced until the RTI responds with a *Time Advance Grant †* service invocation at that joined federate. The interval between these service invocations is defined to be the Time Advancing state; this state is shown in the state chart in Figure 17.

[7]In the case of a nonconstrained joined federate, messages that may be received by that joined federate and that have a sent message order type of TSO may or may not be included in the calculation of the joined federate's LITS.

Each service shall take a specific logical time t_R (time requested) as an argument, shall request slightly different coordination from the RTI, and is further detailed in the service descriptions and summarized in Table 3. The first column represents the form of time advancement that is requested. The second column describes what timestamps a joined federate cannot assign to messages it sends while in the Time Advancing state (i.e., after requesting an advance to t_R, but before the advance is granted). The abbreviation *ts* stands for the timestamp of a message. For example, the entry for *Time Advance Request* with zero lookahead reads "Can't send $ts \leq t_R$." In other words, the joined federate shall not send any TSO messages that have a timestamp less than or equal to the federation time requested by the joined federate. The third column summarizes what messages the RTI shall guarantee to deliver to the advancing joined federate before granting the joined federate's request for time advancement; only messages described in this column shall be delivered. The fourth column is similar to the second, but it shows what constraints apply when the joined federate is in the Time Granted state (i.e., after an advance has been granted) and it has reached the Time Granted state by means of the specified form of time advancement.[8] Constraints in the fourth column are expressed in terms of the logical time (*LT*) of the joined federate. Lookahead in the second and fourth columns refers to the joined federate's current value of lookahead when in the respective states. Since a joined federate's lookahead can change while it is in the Time Granted state, the constraints expressed in this table change accordingly.

Table 3—Service descriptions

Time service	Constraint when in Time Advancing state (t_R is the requested logical time; lookahead refers to the actual lookahead the federate would have if it were granted to t_R)	Messages delivered before advance is granted	Constraint when in the Time Granted state (*LT* is the logical time of the joined federate; lookahead refers to the actual lookahead of the federate while in the Time Granted state)
Time Advance Request	Can't send $ts < t_R$ + lookahead	All queued RO messages All TSO messages with $ts \leq t_R$	Can't send $ts < LT$ + lookahead
Time Advance Request (zero lookahead)	Can't send $ts \leq t_R$	All queued RO messages All TSO messages with $ts \leq t_R$	Can't send $ts \leq LT$
Time Advance Request Available	Can't send $ts < t_R$ + lookahead	All queued RO messages All TSO messages with $ts < t_R$ All queued TSO messages with $ts = t_R$	Can't send $ts < LT$ + lookahead
Next Message Request	Can't send $ts < t_R$ + lookahead	All queued RO messages TSO message with the smallest timestamp that will ever be received in a TSO message and for which $ts \leq t_R$ All other TSO messages with the same *ts*	Can't send $ts < LT$ + lookahead
Next Message Request (zero lookahead)	Can't send $ts \leq t_R$	All queued RO messages TSO message with the smallest timestamp that will ever be received in a TSO message and for which $ts \leq t_R$ All other TSO messages with the same *ts*	Can't send $ts \leq LT$

[8]When time-regulation becomes enabled at a joined federate, that joined federate is treated as if it had advanced its logical time via the *Time Advance Request Available* service.

Table 3—Service descriptions *(continued)*

Time service	Constraint when in Time Advancing state (t_R is the requested logical time; lookahead refers to the actual lookahead the federate would have if it were granted to t_R)	Messages delivered before advance is granted	Constraint when in the Time Granted state (LT is the logical time of the joined federate; lookahead refers to the actual lookahead of the federate while in the Time Granted state)
Next Message Request Available	Can't send $ts < t_R +$ lookahead	All queued RO messages TSO message with the smallest timestamp that will ever be received in a TSO message and for which $ts \le t_R$ All other queued TSO messages with the same *ts*	Can't send $ts < LT +$ lookahead
Flush Queue Request	Can't send $ts < t_R +$ lookahead	All queued RO messages All queued TSO messages	Can't send $ts < LT +$ lookahead

The *Time Advance Grant †* service shall be used to grant an advance regardless of which form of request was made to advance logical time. This service shall take a logical time as an argument, and this value shall be the joined federate's new logical time. The guarantee that the RTI makes about message delivery relative to the provided logical time shall depend on the type of request to advance time; the specific guarantees are provided in the service descriptions. Note that in some cases (i.e., when using *Next Message Request*, *Next Message Request Available*, or *Flush Queue Request* service), the RTI may advance a joined federate to a logical time that is less than the logical time that the joined federate requested (t_R). In other cases (i.e., when using *Time Advance Request* and *Time Advance Request Available* services), the RTI shall advance a joined federate only to the logical time that was requested.

The RTI shall grant an advance to logical time *LT* only when it can guarantee that all TSO messages with timestamps less than *LT* (or in some cases, less than or equal to *LT*) have been delivered to the joined federate. This guarantee enables the joined federate to model the behavior of the entities it represents up to *LT* without concern for receiving new messages with timestamps less than *LT*. Note that in some cases, providing this guarantee shall require the RTI to wait for a significant period of wall-clock time to elapse before it can grant a time advancement to a time-constrained joined federate. However, in the case of joined federates that are not time-constrained (and, thus, cannot receive TSO messages), the guarantee is trivially true, and the advance can be granted almost immediately.

The advancement of logical time by time-regulating joined federates is important because it acts as their guarantee not to send any TSO messages with timestamps less than some specified federation time. In general, when time-regulating joined federates move their logical times forward, time-constrained joined federates can move forward as well.

Joined federates that are not time-regulating need not advance their logical time, but they may do so. Such advancements shall have no effect on other joined federates' time advancement unless the advancing joined federate subsequently becomes time-regulating (at which point the advancing joined federate may begin to have an effect on the advancement of time-constrained joined federates).

8.1.7 Putting it all together

The state chart shown in Figure 17 illustrates when a joined federate may become time-regulating and time-constrained, when time advances may be requested, how a joined federate enables or disables

asynchronous message delivery, the effect these activities have on determining sent and received message order types, and when messages may be sent and received.

The transition labeled “Send Message” shall represent any service invocation that is called sending a message. As represented in the state chart, such a transition can occur in any state and shall result in the joined federate returning to whatever state it was in before the transition. The column to the right of the state chart elaborates on how the order type of the sent message is determined. Each part of the definition of “Send Message” shall be composed of a conversion rule (denoted as two terms separated by an arrow) and an optional boolean guard (denoted in square braces, just as in state charts). The term to the left of the arrow in each conversion rule represents the preferred order type of the message and whether a timestamp was provided by the invoking joined federate. The term to the right of the arrow represents the order type of the sent message. The guard represents under what circumstances the conversion rule applies. Therefore, each part of the definition reads as follows: if the preferred order type of the message is as indicated to the left of the arrow, the usage of a timestamp is as described to the left of the arrow, and the boolean guard (if present) is true, then the order type of the sent message is as indicated to the right of the arrow. The conversion rules provided in the state chart are the same as the results contained in Table 1 and Table 2.

The transitions labeled “Receive Message #1” and “Receive Message #2” read similarly with one exception: The conversion rules are slightly different. The term to the left of the arrow represents the order type of the received message. The term to the right of the arrow represents the order type of the corresponding sent message.

A joined federate may send messages while in any state in this diagram. If the joined federate is time-regulating and sending a TSO message, the timestamp of that message shall be constrained as described in 8.1.4 with one exception: When a joined federate is in the Time Advancing state, the timestamp of sent TSO messages is constrained by the joined federate’s requested logical time (plus its lookahead) rather than by the joined federate’s current logical time (plus its lookahead). If the joined federate is granted to a logical time that is less than its requested logical time (i.e., the request used the *Next Message Request*, *Next Message Request Available*, or *Flush Queue Request* service), the constraints shall ease upon leaving the Time Advancing state. The constraints on advancing and nonadvancing joined federates are summarized in Table 3.

When joined federates are eligible to receive messages is dependent on several factors. If the joined federate is not time-constrained, it may receive messages while in any state (although only RO messages may be received). If the joined federate is time-constrained, it shall normally receive messages only when in the Time Advancing state. However, joined federates may enable asynchronous message delivery (via the *Enable Asynchronous Delivery* service), which permits them to receive RO messages (but not TSO messages) when not in the Time Advancing state.

Which RO messages will be received when a joined federate is eligible to receive RO messages shall depend only on which messages have been sent that will be received as RO messages by that joined federate. In general, if a joined federate is eligible to receive RO messages, it may receive all RO messages that it has not yet received.

Which TSO messages will be received when a joined federate is eligible to receive TSO messages shall depend on which TSO messages have been sent that will be received as TSO messages, what timestamps the messages have, and what form of time advancement was requested. Precisely which TSO messages will be received shall be defined in each of the different time advancement services.

Because messages are not always eligible for delivery, the RTI shall internally queue pending messages for each joined federate. The RTI shall queue all messages that the joined federate will receive as TSO or RO messages. When messages are finally delivered to the joined federate, they shall be removed from the queue.

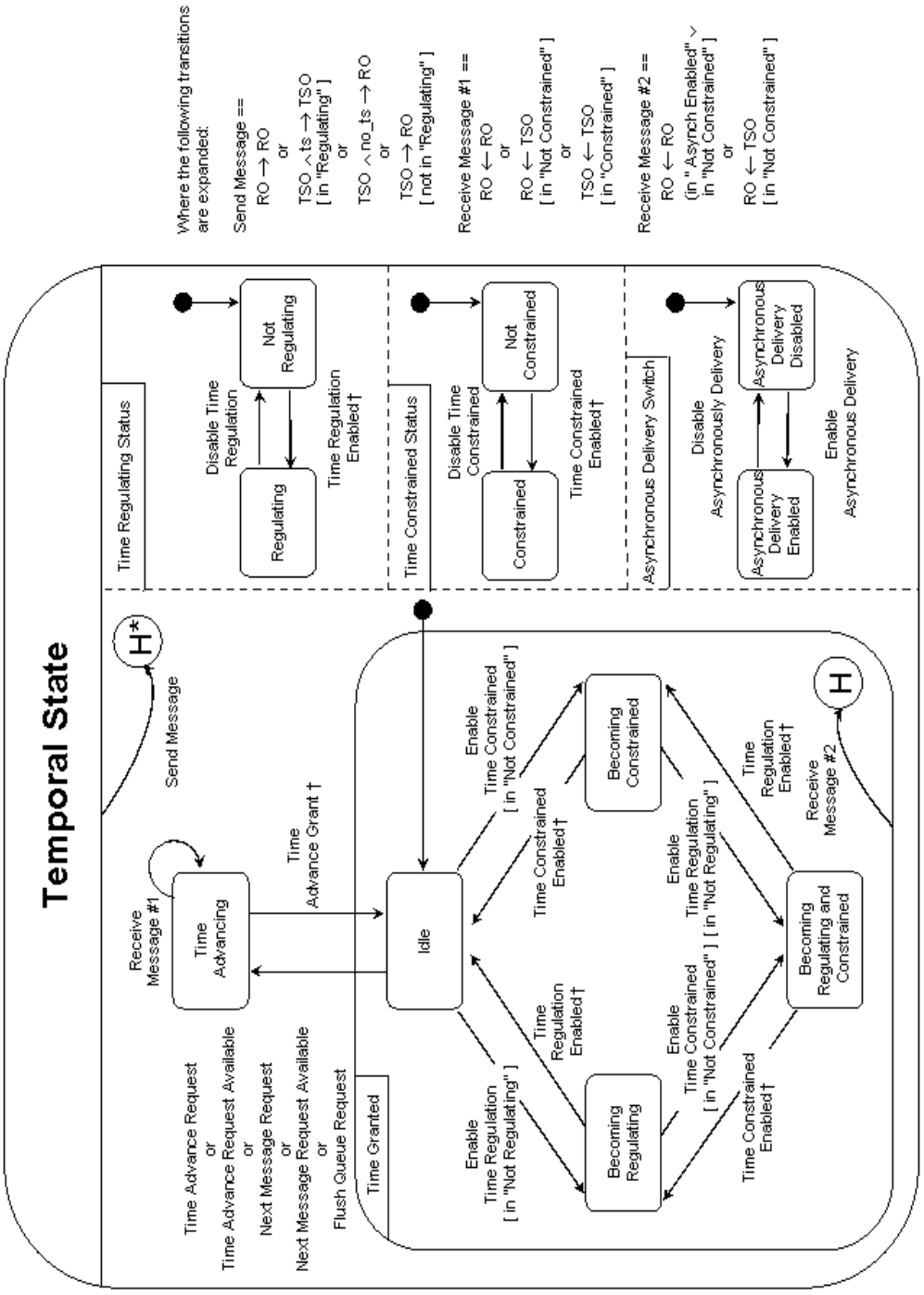

Figure 17—Temporal state

8.1.8 Delay Subscription Evaluation Switch

The federation execution-wide Delay Subscription Evaluation Switch indicates how the RTI shall filter a TSO or RO message based on the receiving joined federate's current subscriptions. The setting for this switch shall come from the FDD at federation execution creation time. This switch setting shall not be modifiable during a federation execution.

Whenever this switch is enabled, the RTI shall delay subscription evaluation of a message until the message is eligible for delivery to the joined federate.

Whenever this switch is disabled, the RTI may determine if a message is eligible for delivery to a joined federate based on the current subscriptions of that joined federate when the message is generated. If this immediate subscription evaluation indicates that the message is not eligible for delivery to the joined federate, then that message shall never be eligible for delivery to the joined federate regardless of any changes to subscriptions (by that joined federate) that occur subsequent to the message generation.

In both cases, when a message is eligible for delivery to a joined federate, the RTI shall use the federate's current subscriptions to determine if it will be actually be delivered to the joined federate.

8.1.9 Warnings on partial use of time management

When there are some joined federates in a federation execution that use time management services and other joined federates that do not, there are some complications of which federation developers should be aware. The complications stem from the fact that some joined federates are making use of TSO ordering of messages, while others are accepting RO ordering of these same messages. The two basic scenarios (which can, of course, occur independently or together in the same federation execution) are as follows:

a) A time-regulating joined federate sends TSO messages that are received by a nonconstrained joined federate in RO.
b) A nonregulating joined federate sends RO messages to a joined federate that expects to receive messages in TSO.

The former tends to be more of a problem because, among other reasons, the time-regulating joined federate need not send its messages in TSO. They may (for example) be received by the nonconstrained joined federate in the same order as sent, which is not equivalent to the order in which the sender meant them to be received.

The latter is not as much of a problem because the nonregulating joined federate will send its messages in the order it means for them to be received. The time-constrained joined federate will receive them in this order (barring factors like differing transportation types among the sent messages).

Another factor to consider is that joined federates do not need to be time-regulating with respect to all messages. A time-regulating joined federate can send some messages TSO and send other messages RO. This situation can also lead to complications if the decision on order type is not carefully considered.

8.1.10 Mixing use of TSO and RO messages

Some specific examples of odd results when mixing the use of TSO and RO messages are presented below:

a) A time-regulating joined federate invokes the *Delete Object Instance* service and provides a timestamp far in its future (much greater than its logical time). Subsequently, it sends several update messages, modeling attributes of the object instance in the interval prior to its deletion and providing a timestamp to each update that is less than the timestamp of the deletion. Nonconstrained joined federates may receive these messages in the order in which they were sent. In other words, a

nonconstrained joined federate may receive the delete message and never receive any of the update messages.

b) A time-regulating joined federate invokes the *Delete Object Instance* service and provides a timestamp. This TSO message will be received as RO by any joined federates that are not time-constrained. This situation may result in nonconstrained joined federates receiving this message much sooner in wall-clock time than time-constrained joined federates if the timestamp provided with the delete was greater than the logical time of most time-constrained joined federates. Even more problematic is if the nonconstrained joined federate later decides to become time-constrained. If it becomes time-constrained at a federation time less than the timestamp of the delete, it may now discover the object instance for which it just received a delete message. However, any messages regarding this object instance may have been dequeued for this joined federate when it received the delete message, and subsequent behavior may not be what is expected.

c) A time-regulating joined federate that is sending both TSO and RO messages can in fact send both kinds of messages in a single *Update Attribute Values* service invocation. The message retraction designator that is returned on this service invocation, however, applies only to the resulting sent TSO messages, not to the whole service invocation. Consequently, any instance attribute values that resulted in sent RO messages shall not be retracted by invocations of the *Retract* service that include the returned message retraction designator.

8.1.11 Order types

The following order types shall be the only ones supported by the RTI:

a) TSO: (legal name shall be “TimeStamp”)
b) RO: (legal name shall be “Receive”)

8.2 *Enable Time Regulation* service

The *Enable Time Regulation* service shall enable time-regulation for the joined federate invoking the service and thereby enable the joined federate to send TSO messages. The joined federate shall request that its lookahead be set to the specified value. The RTI shall indicate the logical time assigned to the joined federate through the *Time Regulation Enabled †* service. The logical time provided when time-regulation is enabled shall be the smallest possible logical time that is greater than or equal to the joined federate’s current logical time and for which all other constraints necessary to ensure TSO message delivery are satisfied. In other words, in general, the logical time that the joined federate will be given (plus the requested lookahead) will always be greater than or equal to the maximum logical time of all time-constrained joined federates (with the possibility of being of equal value depending on what form of time advancement was used by each time-constrained joined federate).

Upon the RTI’s invocation of the corresponding *Time Regulation Enabled †* service, the invoking joined federate may begin sending TSO messages that have a timestamp greater than or equal to the joined federate’s logical time plus the joined federate’s lookahead. Zero lookahead joined federates are not subject to additional restrictions when time-regulation is first enabled.

Because the invocation of this service may require the RTI to advance the invoking joined federate’s logical time, this service has an additional meaning for time-constrained joined federates. Since the advancing logical time for a time-constrained joined federate is synonymous with a guarantee that all TSO messages with timestamps less than the new logical time have been delivered, the invocation of this service shall be considered an implicit *Time Advance Request Available* service invocation. The subsequent invocation of *Time Regulation Enabled †* service shall be considered an implicit *Time Advance Grant †* service invocation. Thus, if a time-constrained joined federate attempts to become time-regulating, it may receive RO and TSO messages between its invocation of the *Enable Time Regulation* service and the RTI’s

invocation of the *Time Regulation Enabled †* service at the joined federate. This special case is not illustrated in the state chart in Figure 17.

8.2.1 Supplied arguments

a) Lookahead.

8.2.2 Returned arguments

a) None.

8.2.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The joined federate is not in the Time Advancing state.
e) The joined federate does not have a request to enable time-regulation pending.
f) Time-regulation is not enabled in the joined federate.
g) Federate save not in progress.
h) Federate restore not in progress.

8.2.4 Postconditions

a) The RTI is informed of the joined federate's request to enable time-regulation (the request is now pending).

8.2.5 Exceptions

a) Invalid lookahead.
b) Joined federate in Time Advancing state.
c) Request for time-regulation pending.
d) Time-regulation is already enabled.
e) Federate save in progress.
f) Federate restore in progress.
g) The federate is not a federation execution member.
h) Not connected.
i) RTI internal error.

8.2.6 Reference state charts

a) Figure 17.

8.3 *Time Regulation Enabled †* service

Invocation of the *Time Regulation Enabled †* service shall indicate that a prior request to enable time regulation has been honored. The value of this service's argument shall indicate that the logical time of the joined federate has been set to the specified value.

8.3.1 Supplied arguments

a) Current logical time of the joined federate.

8.3.2 Returned arguments

a) None.

8.3.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate has a request to enable time-regulation pending.
d) Federate save not in progress.
e) Federate restore not in progress.

8.3.4 Postconditions

a) The joined federate is notified that time-regulation is enabled (the request is no longer pending), and the joined federate may now send TSO messages. The joined federate's logical time shall be set to the value specified as the argument to this service. The joined federate's actual lookahead shall be set to that specified in the corresponding *Enable Time Regulation* request.
b) If the joined federate is time-constrained, no additional TSO messages shall be delivered with timestamps less than the provided federation time.

8.3.5 Exceptions

a) Federate internal error.

8.3.6 Reference state charts

a) Figure 17.

8.4 *Disable Time Regulation* service

Invocation of the *Disable Time Regulation* service shall indicate that the joined federate is disabling time regulation. Subsequent messages sent by the joined federate shall be sent automatically as RO messages.

8.4.1 Supplied arguments

a) None.

8.4.2 Returned arguments

a) None.

8.4.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) Time-regulation is enabled at the joined federate.
e) Federate save not in progress.
f) Federate restore not in progress.

8.4.4 Postconditions

a) The joined federate is no longer time-regulating and may no longer send TSO messages.

8.4.5 Exceptions

a) Time-regulation was not enabled.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

8.4.6 Reference state charts

a) Figure 17.

8.5 *Enable Time Constrained* service

The *Enable Time Constrained* service shall request that the joined federate invoking the service become time-constrained. The RTI shall indicate that the joined federate is time-constrained by invoking the *Time Constrained Enabled †* service.

8.5.1 Supplied arguments

a) None.

8.5.2 Returned arguments

a) None.

8.5.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The joined federate is not in the Time Advancing state.
e) The joined federate does not have a request to enable time-constrained pending.
f) The joined federate is not already time-constrained.
g) Federate save not in progress.
h) Federate restore not in progress.

8.5.4 Postconditions

a) The RTI is informed of the joined federate's request to become time-constrained (the request is now pending).

8.5.5 Exceptions

a) Joined federate in Time Advancing state.
b) Request for time-constrained pending.
c) Time-constrained is already enabled.
d) Federate save in progress.

e) Federate restore in progress.
f) The federate is not a federation execution member.
g) Not connected.
h) RTI internal error.

8.5.6 Reference state charts

a) Figure 17.

8.6 *Time Constrained Enabled †* service

Invocation of the *Time Constrained Enabled †* service shall indicate that a prior request to become time-constrained has been honored. The value of this service's argument shall indicate the new logical time of the joined federate. If the joined federate is time-regulating, the argument shall equal the joined federate's current logical time. The RTI shall provide the joined federate with the smallest possible logical time that is greater than or equal to the logical time of the joined federate and for which all other constraints necessary to ensure TSO message delivery are satisfied. In other words, in general, the provided federation time must be less than or equal to the joined federate's GALT. If the joined federate's current logical time is greater than its GALT, then the joined federate shall not become time-constrained until the joined federate's GALT advances beyond its logical time. See 8.1.6 for more explanation on this point.

When a joined federate changes to be time-constrained, TSO messages stored in the RTI's internal queues that have timestamps greater than or equal to the joined federate's logical time shall be delivered in TSO. TSO messages delivered to the joined federate before it becomes time-constrained, possibly including messages with timestamps greater than or equal to the joined federate's current logical time, shall be delivered as RO messages.

Joined federates that are time-constrained may receive messages only when in the Time Advancing state unless asynchronous message delivery is enabled (by use of the *Enable Asynchronous Delivery †* service). If asynchronous message delivery is enabled, the time-constrained joined federate may receive RO messages when not in the Time Advancing state, but TSO messages may still be received only when in the Time Advancing state.

8.6.1 Supplied arguments

a) Current logical time of the joined federate.

8.6.2 Returned arguments

a) None.

8.6.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate has a request to enable time-constrained pending.
d) Federate save not in progress.
e) Federate restore not in progress.

8.6.4 Postconditions

a) The joined federate has been notified that it is time-constrained.

b) The joined federate has been notified that it may now receive TSO messages (the request is no longer pending), and its logical time advances are constrained so that the joined federate's logical time shall never exceed the GALT computed by the RTI for the joined federate. The joined federate's logical time shall be set to the value specified as the argument to this service.

8.6.5 Exceptions

a) Federate internal error.

8.6.6 Reference state charts

a) Figure 17.

8.7 *Disable Time Constrained* service

Invocation of the *Disable Time Constrained* service shall indicate that the joined federate is no longer time-constrained. All enqueued and subsequent TSO messages shall be delivered to the joined federate as RO messages.

8.7.1 Supplied arguments

a) None.

8.7.2 Returned arguments

a) None.

8.7.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The joined federate is time-constrained.
e) Federate save not in progress.
f) Federate restore not in progress.

8.7.4 Postconditions

a) The joined federate is no longer time-constrained and can no longer receive TSO messages.

8.7.5 Exceptions

a) Time-constrained is not enabled.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

8.7.6 Reference state charts

a) Figure 17.

8.8 *Time Advance Request* service

The *Time Advance Request* service shall request an advance of the joined federate's logical time and release zero or more messages for delivery to the joined federate.

Invocation of this service shall cause the following set of messages to be delivered to the joined federate:

— All messages queued in the RTI that the joined federate will receive as RO messages
— All messages that the joined federate will receive as TSO messages that have timestamps less than or equal to the specified logical time

After invoking the *Time Advance Request* service, the messages shall be passed to the joined federate when the RTI invokes the *Receive Interaction †*, *Reflect Attribute Values †*, and *Remove Object Instance †* services.

By invoking the *Time Advance Request* service with the specified logical time, the joined federate is guaranteeing that it will not generate a TSO message at any point in the future with a timestamp less than or equal to the specified logical time, even if the joined federate's lookahead is zero. Further, the joined federate may not generate any TSO messages in the future with timestamps less than the specified logical time plus the actual lookahead the joined federate would have if it were granted to the specified logical time.

A *Time Advance Grant †* service invocation shall complete this request and indicate to the joined federate that it has advanced to the specified logical time and that no additional TSO messages will be delivered to the joined federate in the future with timestamps less than or equal to the logical time of the grant. For time-constrained joined federates, requests for which the specified logical time is less than GALT can be granted without waiting for other joined federates to advance.

8.8.1 Supplied arguments

a) Logical time.

8.8.2 Returned arguments

a) None.

8.8.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified logical time is greater than or equal to the joined federate's logical time.
e) The joined federate is not in the Time Advancing state.
f) The joined federate does not have a request to enable time-regulation pending.
g) The joined federate does not have a request to enable time-constrained pending.
h) Federate save not in progress.
i) Federate restore not in progress.

8.8.4 Postconditions

a) The joined federate may not send any TSO messages with timestamps less than the specified logical time plus its actual lookahead.
b) If the joined federate's actual lookahead is zero, it may not send any TSO messages with timestamps less than or equal to the specified logical time.
c) The RTI is informed of the joined federate's request to advance its logical time.

8.8.5 Exceptions

a) Logical time already passed.
b) The logical time is invalid.
c) Joined federate in Time Advancing state.
d) Request for time-regulation pending.
e) Request for time-constrained pending.
f) Federate save in progress.
g) Federate restore in progress.
h) The federate is not a federation execution member.
i) Not connected.
j) RTI internal error.

8.8.6 Reference state charts

a) Figure 17.

8.9 *Time Advance Request Available* service

The *Time Advance Request Available* service shall request an advance of the joined federate's logical time. It is similar to the *Time Advance Request* service to logical time *T,* except

— The RTI shall not guarantee delivery of all messages with timestamps equal to *T* when a *Time Advance Grant †* service invocation to logical time *T* is issued
— After the joined federate receives a *Time Advance Grant †* service invocation to logical time *T*, it can send additional messages with timestamps equal to *T* if the joined federate's actual lookahead is zero

Invocation of this service shall cause the following set of messages to be delivered to the joined federate:

— All messages queued in the RTI that the joined federate will receive as RO messages
— All messages that the joined federate will receive as TSO messages that have timestamps less than the specified logical time
— Any messages queued in the RTI that the joined federate will receive as TSO messages that have timestamps equal to the specified logical time

After invoking the *Time Advance Request Available* service, the messages shall be passed to the joined federate when the RTI invokes the *Receive Interaction †, Reflect Attribute Values †,* and *Remove Object Instance †* services.

By invoking the *Time Advance Request Available* service with the specified logical time, the joined federate is guaranteeing that it will not generate a TSO message at any point in the future with a timestamp less than the specified logical time plus the actual lookahead the joined federate would have if it were granted to the specified logical time.

A *Time Advance Grant †* service invocation shall complete this request and indicate to the joined federate that it has advanced to the specified logical time and no additional TSO messages shall be delivered to the joined federate in the future with timestamps less than the logical time of the grant. Additional messages with timestamps equal to the logical time of the grant can arrive in the future. For time-constrained joined federates, requests for which the specified logical time is less than or equal to GALT can be granted without waiting for other joined federates to advance.

8.9.1 Supplied arguments

a) Logical time.

8.9.2 Returned arguments

a) None.

8.9.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified logical time is greater than or equal to the joined federate's logical time.
e) The joined federate is not in the Time Advancing state.
f) The joined federate does not have a request to enable time-regulation pending.
g) The joined federate does not have a request to enable time-constrained pending.
h) Federate save not in progress.
i) Federate restore not in progress.

8.9.4 Postconditions

a) The joined federate may not send any TSO messages with timestamps less than the specified logical time plus its actual lookahead.
b) The RTI is informed of the joined federate's request to advance logical time.

8.9.5 Exceptions

a) Logical time has already passed.
b) The logical time is invalid.
c) Joined federate in Time Advancing state.
d) Request for time-regulation pending.
e) Request for time-constrained pending.
f) Federate save in progress.
g) Federate restore in progress.
h) The federate is not a federation execution member.
i) Not connected.
j) RTI internal error.

8.9.6 Reference state charts

a) Figure 17.

8.10 *Next Message Request* service

The *Next Message Request* service shall request the logical time of the joined federate to be advanced to the timestamp of the next TSO message that will be delivered to the joined federate, provided that message has a timestamp no greater than the logical time specified in the request.

Invocation of this service shall cause the following set of messages to be delivered to the joined federate:

— All messages queued in the RTI that the joined federate will receive as RO messages

— The smallest timestamped message that will ever be received by the joined federate as a TSO message with a timestamp less than or equal to the specified logical time, and all other messages containing the same timestamp that the joined federate will receive as TSO messages

After invocation of the *Next Message Request* service, the messages shall be passed to the joined federate when the RTI invokes the *Receive Interaction †, Reflect Attribute Values †,* and *Remove Object Instance †* services.

By invoking *Next Message Request* with the specified logical time, the joined federate is guaranteeing that it will not generate a TSO message before the next *Time Advance Grant †* service invocation with a timestamp less than or equal to the specified logical time (or less than the specified logical time plus the actual lookahead the joined federate would have if it were granted to the specified logical time if its lookahead is not zero).

If it does not receive any TSO messages before the *Time Advance Grant †* service invocation, the joined federate shall guarantee that it will not generate a TSO message at any point in the future with a timestamp less than or equal to the specified logical time (or less than the specified logical time plus its actual lookahead if its lookahead is not zero).

If it does receive any TSO messages before the *Time Advance Grant †* service invocation, the joined federate shall guarantee that it will not generate a TSO message at any point in the future with a timestamp less than or equal to the logical time of the grant (or less than the logical time of the grant plus its actual lookahead if its lookahead is not zero).

A *Time Advance Grant †* service invocation shall complete this request and indicate to the joined federate that it has advanced its logical time to the timestamp of the TSO messages that are delivered, if any, or to the specified logical time if no TSO messages were delivered. It shall also indicate that no TSO messages will be delivered to the joined federate in the future with timestamps less than or equal to the logical time of the grant. For time-constrained joined federates, requests for which either LITS or the specified logical time is less than GALT can be granted without waiting for other joined federates to advance.

8.10.1 Supplied arguments

a) Logical time.

8.10.2 Returned arguments

a) None.

8.10.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified logical time is greater than or equal to the joined federate's logical time.
e) The joined federate is not in the Time Advancing state.
f) The joined federate does not have a request to enable time-regulation pending.
g) The joined federate does not have a request to enable time-constrained pending.
h) Federate save not in progress.
i) Federate restore not in progress.

8.10.4 Postconditions

a) The joined federate may not send any TSO messages with timestamps less than the specified logical time plus its actual lookahead.
b) If the joined federate's lookahead is zero, it may not send any TSO messages with timestamps less than or equal to the specified logical time.
c) The RTI is informed of the joined federate's request to advance logical time.

8.10.5 Exceptions

a) Logical time has already passed.
b) The logical time is invalid.
c) Joined federate in Time Advancing state.
d) Request for time-regulation pending.
e) Request for time-constrained pending.
f) Federate save in progress.
g) Federate restore in progress.
h) The federate is not a federation execution member.
i) Not connected.
j) RTI internal error.

8.10.6 Reference state charts

a) Figure 17.

8.11 *Next Message Request Available* service

The *Next Message Request Available* service shall request the logical time of the joined federate to be advanced to the timestamp of the next TSO message that will be delivered to the joined federate, provided that message has a timestamp no greater than the logical time specified in the request. It is similar to the *Next Message Request* service except

— The RTI shall not guarantee delivery of all messages with timestamps equal to *T* when a *Time Advance Grant †* service invocation to logical time *T* is issued, and
— After the joined federate receives a *Time Advance Grant †* service invocation to logical time *T*, it can send additional messages with timestamps equal to *T* if the joined federate's lookahead is zero.

Invocation of this service shall cause the following set of messages to be delivered to the joined federate:

— All messages queued in the RTI that the joined federate will receive as RO messages
— The smallest timestamped message that will ever be received by the joined federate as a TSO message with a timestamp less than or equal to the specified logical time, and any other messages queued in the RTI that the joined federate will receive as TSO messages and that have the same timestamp.

After invoking the *Next Message Request Available* service, the messages shall be passed to the joined federate when the RTI invokes the *Receive Interaction †, Reflect Attribute Values †,* and *Remove Object Instance †* services.

By invoking the *Next Message Request Available* service with the specified logical time, the joined federate is guaranteeing that it will not generate a TSO message before the next *Time Advance Grant †* service invocation with a timestamp less than the specified logical time plus the actual lookahead the joined federate would have if it were granted to the specified logical time.

If it does not receive any TSO messages before the *Time Advance Grant †* service invocation, the joined federate shall guarantee that it will not generate a TSO message at any point in the future with a timestamp less than the specified logical time plus its actual lookahead.

If it does receive any TSO messages before the *Time Advance Grant †* service invocation, the joined federate shall guarantee that it will not generate a TSO message at any point in the future with a timestamp less than the logical time of the grant plus its actual lookahead.

A *Time Advance Grant †* service invocation shall complete this request and indicate to the joined federate that it has advanced its logical time to the timestamp of the TSO messages that are delivered, if any, or to the specified logical time if no TSO messages were delivered. Even if no TSO messages were delivered, it is still possible that TSO messages may be delivered in the future with timestamps equal to the logical time of the grant. A *Time Advance Grant †* service invocation shall also indicate that no TSO messages will be delivered to the joined federate in the future with timestamps less than the logical time of the grant. For time-constrained joined federates, requests for which LITS or the specified logical time is less than or equal to GALT can be granted without waiting for other joined federates to advance.

8.11.1 Supplied arguments

a) Logical time.

8.11.2 Returned arguments

a) None.

8.11.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified logical time is greater than or equal to the joined federate's logical time.
e) The joined federate is not in the Time Advancing state.
f) The joined federate does not have a request to enable time-regulation pending.
g) The joined federate does not have a request to enable time-constrained pending.
h) Federate save not in progress.
i) Federate restore not in progress.

8.11.4 Postconditions

a) The joined federate may not send TSO messages with timestamps less than the specified logical time plus its actual lookahead.
b) The RTI is informed of the joined federate's request to advance logical time.

8.11.5 Exceptions

a) Logical time has already passed.
b) The logical time is invalid.
c) Joined federate in Time Advancing state.
d) Request for time-regulation pending.
e) Request for time-constrained pending.
f) Federate save in progress.
g) Federate restore in progress.
h) The federate is not a federation execution member.
i) Not connected.

j) RTI internal error.

8.11.6 Reference state charts

a) Figure 17.

8.12 *Flush Queue Request* service

The *Flush Queue Request* service shall request that all messages queued in the RTI that the joined federate will receive as TSO messages be delivered now. The RTI shall deliver all such messages as soon as possible, despite the fact that it may not be able to guarantee that no future messages containing smaller timestamps could arrive. 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 timestamp of all TSO messages delivered by the RTI in response to this invocation of the *Flush Queue Request* service

Invocation of this service shall cause the following set of messages to be delivered to the joined federate:

— All messages queued in the RTI that the joined federate will receive as RO messages
— All messages queued in the RTI that the joined federate will receive as TSO messages

After invoking the *Flush Queue Request* service, the messages shall be passed to the joined federate when the RTI invokes the *Receive Interaction †, Reflect Attribute Values †,* and *Remove Object Instance †* services.

By invoking the *Flush Queue Request* service with the specified logical time, the joined federate is guaranteeing that it will not generate a TSO message before the next *Time Advance Grant †* service invocation with a timestamp less than the specified logical time plus the actual lookahead the joined federate would have if it were granted to the specified logical time.

After the *Time Advance Grant †* service invocation, the joined federate shall guarantee that it shall not generate a TSO message at any point in the future with a timestamp less than the logical time of the grant plus its actual lookahead.

A *Time Advance Grant †* service invocation shall complete this request and indicate to the joined federate that it has advanced to the logical time of the grant and no additional TSO messages shall be delivered to the joined federate in the future with timestamps less than the logical time of the grant. A *Flush Queue Request* service can always be granted without waiting for other joined federates to advance.

Since the *Flush Queue Request* service results in the delivery of all queued TSO messages, not just those the RTI can guarantee are safe to deliver, there are two consequences the receiving joined federate must be prepared to handle. First, messages may later be received that have timestamps that are smaller than those of some messages received in response to the *Flush Queue Request* service invocation, and this sequencing violates timestamped ordering. Second, some messages delivered may still be eligible for retraction by their originators; therefore, the receiving joined federate must be capable of dealing with *Request Retraction †* service invocations for these messages.

8.12.1 Supplied arguments

a) Logical time.

8.12.2 Returned arguments

a) None.

8.12.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified logical time is greater than or equal to the joined federate's logical time.
e) The joined federate is not in the Time Advancing state.
f) The joined federate does not have a request to enable time-regulation pending.
g) The joined federate does not have a request to enable time-constrained pending.
h) Federate save not in progress.
i) Federate restore not in progress.

8.12.4 Postconditions

a) The joined federate may not send any TSO messages with timestamps less than the specified logical time plus its actual lookahead.
b) The RTI is informed of the joined federate's request to advance logical time.

8.12.5 Exceptions

a) Logical time has already passed.
b) The logical time is invalid.
c) Joined federate in Time Advancing state.
d) Request for time-regulation pending.
e) Request for time-constrained pending.
f) Federate save in progress.
g) Federate restore in progress.
h) The federate is not a federation execution member.
i) Not connected.
j) RTI internal error.

8.12.6 Reference state charts

a) Figure 17.

8.13 *Time Advance Grant †* service

Invocation of the *Time Advance Grant †* service shall indicate that a prior request to advance the joined federate's logical time has been honored. The argument of this service shall indicate that the logical time for the joined federate has been advanced to this value.

If the grant is issued in response to invocation of a *Next Message Request* or *Time Advance Request* service, the RTI shall guarantee that no additional TSO messages shall be delivered in the future with timestamps less than or equal to this value.

If the grant is in response to an invocation of a *Time Advance Request Available*, *Next Message Request Available*, or *Flush Queue Request* service, the RTI shall guarantee that no additional TSO messages shall be delivered in the future with timestamps less than the value of the grant.

8.13.1 Supplied arguments

a) Logical time.

8.13.2 Returned arguments

a) None.

8.13.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The joined federate is in the Time Advancing state.
d) Federate save not in progress.
e) Federate restore not in progress.

8.13.4 Postconditions

a) If the joined federate has a change to its lookahead pending, its new actual lookahead shall be equal to the maximum of its requested lookahead and its old actual lookahead less the amount of logical time advanced (the joined federate's old logical time less the supplied logical time).
b) The joined federate may not send TSO messages with timestamps less than the supplied logical time plus its actual lookahead.
c) If this service is invoked in response to a *Time Advance Request* or *Next Message Request* service, the joined federate's actual lookahead is zero, and the joined federate may not send TSO messages with timestamps less than or equal to the supplied logical time.
d) If this service is invoked in response to a *Time Advance Request* or *Next Message Request* service, no additional TSO messages shall be delivered with timestamps less than or equal to the supplied logical time.
e) If this service is invoked in response to a *Time Advance Request Available*, *Next Message Request Available*, or *Flush Queue Request* service, no additional TSO messages shall be delivered with timestamps less than the supplied logical time.

8.13.5 Exceptions

a) Federate internal error.

8.13.6 Reference state charts

a) Figure 17.

8.14 *Enable Asynchronous Delivery* service

Invocations of the *Enable Asynchronous Delivery* service shall instruct the RTI to deliver received RO messages to the invoking joined federate when it is in either the Time Advancing state or Time Granted state.

8.14.1 Supplied arguments

a) None.

8.14.2 Returned arguments

a) None.

8.14.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) Asynchronous delivery is disabled at the joined federate.
e) Federate save not in progress.
f) Federate restore not in progress.

8.14.4 Postconditions

a) Asynchronous delivery is enabled at the joined federate.

8.14.5 Exceptions

a) Asynchronous delivery is already enabled.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

8.14.6 Reference state charts

a) Figure 17

8.15 *Disable Asynchronous Delivery* service

Invocations of the *Disable Asynchronous Delivery* service shall, for a time-constrained federate, instruct the RTI to deliver RO messages to the invoking federate only when it is in the Time Advancing state.

8.15.1 Supplied arguments

a) None.

8.15.2 Returned arguments

a) None.

8.15.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) Asynchronous delivery is enabled at the joined federate.
e) Federate save not in progress.
f) Federate restore not in progress.

8.15.4 Postconditions

a) Asynchronous delivery is disabled at the joined federate.

8.15.5 Exceptions

a) Asynchronous delivery is already disabled.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

8.15.6 Reference state charts

a) Figure 17.

8.16 *Query GALT* service

The *Query GALT* service shall request the invoking joined federate's current GALT. The first returned argument shall indicate whether GALT is defined for the joined federate. If the argument indicates that GALT is defined for the joined federate, then the optional returned argument shall be supplied to indicate the joined federate's current GALT.

8.16.1 Supplied arguments

a) None.

8.16.2 Returned arguments

a) GALT definition indicator.
b) Optional current value of invoking joined federate's GALT.

8.16.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) Federate save not in progress.
e) Federate restore not in progress.

8.16.4 Postconditions

a) The joined federate receives the current value of its GALT, if defined.

8.16.5 Exceptions

a) Federate save in progress.
b) Federate restore in progress.
c) The federate is not a federation execution member.
d) Not connected.
e) RTI internal error.

8.16.6 Reference state charts

a) None.

8.17 *Query Logical Time* service

The *Query Logical Time* service shall request the current logical time of the invoking joined federate.

8.17.1 Supplied arguments

b) None.

8.17.2 Returned arguments

a) The invoking joined federate's current logical time.

8.17.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) Federate save not in progress.
e) Federate restore not in progress.

8.17.4 Postconditions

a) The joined federate receives the current value of its logical time.

8.17.5 Exceptions

a) Federate save in progress.
b) Federate restore in progress.
c) The federate is not a federation execution member.
d) Not connected.
e) RTI internal error.

8.17.6 Reference state charts

a) None.

8.18 *Query LITS* service

The *Query LITS* service shall request the invoking joined federate's current LITS. The first returned argument shall indicate whether LITS is defined for the joined federate. If the argument indicates that LITS is defined for the joined federate, then the optional returned argument shall be supplied to indicate the joined federate's current LITS.

8.18.1 Supplied arguments

a) None.

8.18.2 Returned arguments

a) LITS definition indicator.
b) Optional current value of invoking joined federate's LITS.

8.18.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) Federate save not in progress.
e) Federate restore not in progress.

8.18.4 Postconditions

a) The joined federate receives its LITS, if defined.

8.18.5 Exceptions

a) Federate save in progress.
b) Federate restore in progress.
c) The federate is not a federation execution member.
d) Not connected.
e) RTI internal error.

8.18.6 Reference state charts

a) None.

8.19 *Modify Lookahead* service

The *Modify Lookahead* service shall request a change to the joined federate's lookahead. A joined federate may not request a change to its lookahead while it is in the Time Advancing state. The specified lookahead shall be greater than or equal to zero. If the requested value is greater than or equal to the joined federate's actual lookahead, the change shall take effect immediately, and the requested lookahead shall become the actual lookahead. If the requested value is less than the joined federate's actual lookahead, the change shall take effect gradually as the joined federate advances its logical time, and the actual lookahead is initially unchanged. Specifically, the joined federate's actual lookahead shall decrease by T units whenever logical time advances T units until the requested lookahead is reached.

8.19.1 Supplied arguments

a) Requested lookahead.

8.19.2 Returned arguments

a) None.

8.19.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.

d) Time-regulation is enabled in the joined federate.
e) The joined federate is not in the Time Advancing state.
f) Federate save not in progress.
g) Federate restore not in progress.

8.19.4 Postconditions

a) If the requested lookahead is greater than or equal to the joined federate's actual lookahead, the joined federate's actual lookahead shall be set to the requested value.
b) If the requested lookahead is less than the joined federate's actual lookahead, the RTI shall be informed of the joined federate's requested lookahead value.

8.19.5 Exceptions

a) Invalid lookahead.
b) Joined federate in Time Advancing state.
c) Time-regulation was not enabled.
d) Federate save in progress.
e) Federate restore in progress.
f) The federate is not a federation execution member.
g) Not connected.
h) RTI internal error.

8.19.6 Reference state charts

a) None.

8.20 *Query Lookahead* service

The *Query Lookahead* service shall query the RTI for the joined federate's current actual lookahead. The current value of actual lookahead may differ temporarily from the requested lookahead given in the *Modify Lookahead* service if the joined federate is attempting to reduce its actual lookahead.

8.20.1 Supplied arguments

a) None.

8.20.2 Returned arguments

a) The invoking joined federate's current actual lookahead.

8.20.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) Time-regulation is enabled in the joined federate.
e) Federate save not in progress.
f) Federate restore not in progress.

8.20.4 Postconditions

a) The joined federate receives the current value of its actual lookahead.

8.20.5 Exceptions

a) Time-regulation was not enabled.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

8.20.6 Reference state charts

a) None.

8.21 *Retract* service

The *Retract* service shall be used by a joined federate to notify the federation execution that a message previously sent by the joined federate is to be retracted. The *Update Attribute Values*, *Send Interaction*, *Send Interaction with Regions*, and *Delete Object Instance* services shall return a message retraction designator that is used to specify the message that is to be retracted. Retracting a message shall cause the invocation of the *Request Retraction †* service in all joined federates that received the original message.

Retracting a *Delete Object Instance* service message shall result in the reconstitution of the corresponding object instance. This action shall cause the ownership reassumption of the attributes of the affected object instance by the joined federates that owned them when the *Delete Object Instance* service was invoked.

A message may be retracted only if all of the following occur:

a) The retracting joined federate sent the message.
b) The message had a sent order type of TSO (i.e., a message retraction designator was returned by the RTI).
c) The joined federate is time-regulating, and either
    1) The joined federate is in the Time Granted state and the message associated with the specified retraction designator contained a timestamp that is larger than the joined federate's current logical time plus its actual lookahead, or
    2) The joined federate is in the Time Advancing state and the message associated with the specified retraction designator contained a timestamp that is larger than the logical time specified in the joined federate's most recent advance request plus the joined federate's actual lookahead.

8.21.1 Supplied arguments

a) Message retraction designator.

8.21.2 Returned arguments

a) None.

8.21.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The joined federate is time-regulating.

e) The joined federate has issued *Update Attribute Values*, *Send Interaction*, *Send Interaction with Regions*, or *Delete Object Instance* service invocations previously and obtained the message retraction designator.
f) If the joined federate is in the Time Granted state, the message associated with the specified retraction designator contained a timestamp that is larger than the joined federate's current logical time plus its actual lookahead.
g) If the joined federate is in the Time Advancing state, the message associated with the specified retraction designator contained a timestamp that is larger than the logical time specified in the joined federate's most recent advance request plus its actual lookahead.
h) Federate save not in progress.
i) Federate restore not in progress.

8.21.4 Postconditions

a) The RTI is informed that the joined federate requests to retract the specified message.

8.21.5 Exceptions

a) The retraction designator is associated with a message that can no longer be retracted.
b) The message retraction designator is invalid.
c) The federate is not time-regulating.
d) Federate save in progress.
e) Federate restore in progress.
f) The federate is not a federation execution member.
g) Not connected.
h) RTI internal error.

8.21.6 Reference state charts

a) None.

8.22 *Request Retraction †* service

If the RTI receives a legal *Retract* service invocation for a message that has already been delivered to a joined federate, the *Request Retraction †* service shall be invoked on that joined federate. If the message in question is queued in the RTI and has not been delivered to the joined federate, the message shall be dequeued from the RTI and will never be delivered to the joined federate.

Time-constrained joined federates that do not use the *Flush Queue Request* service are not subject to invocation of the *Request Retraction †* service on any received TSO message because they will never receive TSO messages that are eligible for retraction. Nonconstrained joined federates, however, must be prepared to deal with invocations of this service because any message received that was sent as a TSO message may be eligible for retraction.

Since the joined federate is required to save the message retraction designator in order to perform a retraction, the RTI is not required to ensure the message was actually delivered to the federate. The RTI is only required to ensure that a message is not delivered to a joined federate after a retraction for that message has been requested.

8.22.1 Supplied arguments

a) Message retraction designator.

8.22.2 Returned arguments

a) None.

8.22.3 Preconditions

a) The federation execution exists.
b) The federate is joined to that federation execution.
c) Federate save not in progress.
d) Federate restore not in progress.

8.22.4 Postconditions

a) The joined federate has been directed to retract the specified message.

8.22.5 Exceptions

a) Federate internal error.

8.22.6 Reference state charts

a) None.

8.23 *Change Attribute Order Type* service

The preferred order type for each attribute of an object instance shall be initialized from the object class description in the FDD. A joined federate may choose to change the preferred order type during execution. Invoking the *Change Attribute Order Type* service shall change the order type for all future *Update Attribute Values* service invocations for the specified instance attributes. When the ownership of an instance attribute is changed, the preferred order type shall revert to that defined in the FDD.

8.23.1 Supplied arguments

a) Object instance designator.
b) Set of attribute designators.
c) Order type.

8.23.2 Returned arguments

a) None.

8.23.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.
e) The joined federate knows about the object instance with the specified designator.
f) The specified class attributes are available attributes of the object instance's known class.
g) The attributes are defined in the FDD.
h) The joined federate owns the instance attributes.
i) The order type is valid (see 8.1.11).
j) Federate save not in progress.

k) Federate restore not in progress.

8.23.4 Postconditions

a) The order type is changed for the specified instance attributes.

8.23.5 Exceptions

a) The joined federate does not own the specified instance attributes.
b) The specified class attributes are not available attributes of the known object class.
c) The object instance is not known.
d) Federate save in progress.
e) Federate restore in progress.
f) The federate is not a federation execution member.
g) Not connected.
h) RTI internal error.

8.23.6 Reference state charts

a) None.

8.24 *Change Interaction Order Type* service

The preferred order type of each interaction shall be initialized from the interaction class description in the FDD. A joined federate may choose to change the preferred order type during execution. Invoking the *Change Interaction Order Type* service shall change the order type for all future *Send Interaction* and *Send Interaction with Regions* service invocations for the specified interaction class for the invoking joined federate only.

8.24.1 Supplied arguments

a) Interaction class designator.
b) Order type.

8.24.2 Returned arguments

a) None.

8.24.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The interaction class is defined in the FDD.
e) The joined federate is publishing the interaction class.
f) The order type is valid (see 8.1.11).
g) Federate save not in progress.
h) Federate restore not in progress.

8.24.4 Postconditions

a) The preferred order type is changed for the specified interaction class.

8.24.5 Exceptions

a) The joined federate is not publishing the interaction class.
b) The interaction class is not defined in the FDD.
c) Federate save in progress.
d) Federate restore in progress.
e) The federate is not a federation execution member.
f) Not connected.
g) RTI internal error.

8.24.6 Reference state charts

a) None.

9. Data distribution management (DDM)

9.1 Overview

DDM services may be used by joined federates to reduce both the transmission and the reception of irrelevant data. Whereas DM services provide information on data relevance at the class attribute level, DDM services add the capability to further refine the data requirements at the instance attribute level. Similarly, whereas DM services provide information on data relevance at the interaction class level, DDM services add the capability to further refine the data requirements at the specific interaction level. Producers and consumers of data communicated between joined federates may use the DDM services to bound the relevance of such communicated data. As a result, the RTI can recognize the irrelevant data and prevent its delivery to consumers. The relevance of communicating interactions or instance attribute updates is expressed by giving a bounding for relevant communication in a space of user-defined dimensions. Both consumers and producers specify the upper and lower bounds for the relevant portion of that space, and the overlap of producers' and consumers' region sets is used to bound relevant communication.

9.1.1 Definitions for DDM

The DDM services shall be based on the following concepts and terms:

a) A **dimension** shall be a named interval of nonnegative integers. The interval shall be defined by an ordered pair of values. The first value is 0 (zero) for every dimension. The second value, which can vary for each dimension, is that dimension's upper bound as specified in the FDD.
b) A **range** shall be a continuous semi-open interval on a dimension. The interval is defined by an ordered pair of values. The first component of the pair shall be called **range lower bound**, and the second component shall be called **range upper bound**. The range upper bound shall be strictly greater than the range lower bound. The minimum possible difference between a range lower bound and a range upper bound shall be 1 (one).
c) A **region specification** shall be a set of ranges. The dimensions contained in a region specification shall be the dimensions of the ranges that are included in the region specification. A region specification shall contain at most one range for any given dimension. Each range in a region specification shall be defined in terms of the bounds [0, the corresponding dimension's upper bound).
d) A **region template** shall be an incomplete region specification where one or more dimensions have not been assigned ranges.
e) A **region realization** shall be a region specification that is associated with an instance attribute for update, with a sent interaction or with a class attribute or interaction class for subscription. The generic term **region** may be used in circumstances where a region specification, a region realization, or both apply.
f) The RTI shall provide the **default region**, which is defined as having the range [0, the range's dimension's upper bound) for each dimension found in the FDD. The default region shall have all dimensions found in the FDD, regardless of the class attribute or interaction class with which it is associated. The default region realization shall not have a region specification. There shall be no way for a joined federate to refer to the default region. If a joined federate creates a region that has as its dimensions all dimensions found in the FDD and has the range [0, the range's dimension's upper bound) for each dimension, this region shall be equivalent to the default region, but it is not the default region.
g) A collection of attribute designator set and region designator set pairs shall be interpreted as a set of (attribute designator, region designator) pairs. The interpreted form can be viewed as if constructed in the following manner:
    1) The interpreted form begins as an empty set of (attribute designator, region designator) pairs.
    2) For each pair (of sets) in the attribute designator set and region designator set pair collection, create the cross product of the attribute designator set with its region designator set. The result

is a collection of (attribute designator, region designator) pairs. The union of all such (attribute designator, region designator) pairs shall be the interpreted form.

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 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.

9.1.2 Relating region specifications and region realizations

The following relationships, established in the FDD, shall pertain to dimensions:

a) The available dimensions of a class attribute shall be the dimensions associated with that class attribute in the FDD. The available dimensions of an instance attribute shall be the available dimensions of the corresponding class attribute.
b) The available dimensions of an interaction class shall be the dimensions associated with that class in the FDD. The available dimensions of a sent interaction shall be the available dimensions of the interaction class specified in the *Send Interaction With Regions* service invocation.
c) DDM services shall not be invoked for class attributes, instance attributes, interaction classes, or sent interactions that do not have available dimensions.
d) Each dimension in the FDD shall have either a default range specified in terms of the bounds [0, the dimension's upper bound) or shall have an **excluded indicator** specified in the Value When Unspecified Field of the FDD Dimension Table. Where a default range is specified for a dimension in the FDD and the dimension is not mentioned in a region specification, it is implicitly added, with the specified default range, when creating a region realization.

9.1.3 Using regions

9.1.3.1 Region relationships

The following relationships, established through DDM services, shall pertain to regions:

a) 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 (see 9.2) to create a region with a specific set of dimensions.
   2) For every dimension that was explicitly specified when that region was created, invoke the *Set Range Bounds* service (see 10.32) to set the lower and upper bounds of the range of that dimension for that region.
   3) Invoke *Commit Region Modifications* (see 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.

b) 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 are created. These region realizations, however, do not have designators.

c) The specified dimensions of the region specification shall be the dimensions that are explicitly provided when the *Create Region* service is invoked to create that region specification. 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 specification are.

d) The unspecified dimensions of a region realization shall be the available dimensions of the class attribute, instance attribute, interaction class, or sent interaction with which the region realization is associated less the specified dimensions of the region specification from which the region realization is **derived**.

e) The *Commit Region Modifications* service can only either
   1) Create a region specification from a region template, or
   2) 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.

f) When a region realization is created by associating a region specification with an instance attribute, a class attribute, an interaction class, or a sent interaction, additions may be made to the region realization by the RTI to account for default dimension values indicated in the FDD as follows:
   1) All specified dimensions of the region specification shall be included in the region realization, and the range associated with each specified dimension in the region realization shall be the range found in the region specification for the same dimension.
   2) For each unspecified dimension of the region realization, if the FDD indicates a default range other than the excluded indicator, this default range shall be added to the region realization.
   3) When a federate modifies a region specification from which region realizations have been derived, the RTI shall fill in unspecified dimensions in the region realizations in this same manner.

g) A region specification may contain an empty set of ranges. The region realization derived from such a region specification shall contain only the dimensions for which the FDD defines a default range and that are available dimensions of the instance attribute, class attribute, interaction class, or sent interaction with which the region specification is being associated. This situation may result in a region realization that contains no dimensions if all available dimensions have the excluded indicator specified in the FDD; such a region realization shall not overlap with any other region realization, even the default region.

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,

h) If a federate invokes the *Create Region* service and then does not invoke the *Set Range Bounds* service or invokes the *Set Range Bounds* service for only some, but not 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 shall take effect 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.

i) 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, not a region specification. The effects of the *Set Range Bounds* service invocations are still pending and shall 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) shall 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 shall become the range values for that dimension of that region specification.

9.1.3.2 Instance attribute relationships

The following relationships, established through DDM services, shall pertain to object classes, class attributes, object instances, and instance attributes:

a) A region realization other than the default region shall be used for update of an instance attribute if the joined federate owns the instance attribute and has used the instance attribute and region specification from which the region realization was derived as arguments either in the *Register Object Instance With Regions* service or in the *Associate Regions For Updates* service.

   The default region shall be used for update of an instance attribute if and only if the corresponding class attribute has available dimensions and there is no other region realization used for update for that instance attribute. This situation may occur if the *Register Object Instance* service is used to register an instance of an object class that has attributes with available dimensions or if a region association for update is removed via the *Unassociate Regions for Updates* service.

Subsequently invoking the *Unassociate Regions For Updates* service for the same (object instance, region, attribute) triple shall cause that region not to be used for update of that instance attribute. Invoking the *Associate Regions For Updates* service or the *Unassociate Regions For Updates* service with an empty set of regions shall make no changes to the set of associations of the instance attribute.

Any region specifications used to generate region realizations associated with an instance attribute for update shall contain only dimensions that are a subset of the available dimensions of that instance attribute. The resulting region realizations shall have the same property.

A joined federate shall use a region for update of an instance attribute to assert properties of that instance attribute. These properties shall be used to limit reflection of the instance attributes when the joined federate invokes the *Update Attribute Values* service. If a region other than the default region is used for update of a particular instance attribute by a joined federate and the joined federate loses ownership of that instance attribute, that region shall no longer be used for update of that instance attribute.

A joined federate may create and modify update region associations with an instance attribute that is owned by another joined federate or that is unowned. These associations shall not take effect until the joined federate making the associations has acquired ownership of the specified instance attribute. These associations shall take effect immediately upon acquiring ownership of the specified instance attributes.

In all cases, if a joined federate divests ownership of an instance attribute with which an update region is associated, that association shall be lost, as if the *Unassociate Regions For Updates* service had been invoked for that (object instance, region, attribute) triple. Update region associations that are lost shall not come into effect if the joined federate subsequently acquires ownership of the instance attribute unless the joined federate explicitly recreates the association.

b) A region realization other than the default region shall be used for subscription of a class attribute if the joined federate has used the class attribute and a given object class and the region specification from which the region realization was derived as arguments in the *Subscribe Object Class Attributes With Regions* service. Subsequently invoking the *Unsubscribe Object Class Attributes With Regions* service for the same (object class, region, attribute) triple shall cause the region not to be used for subscription of that class attribute. Invoking the *Subscribe Object Class Attributes With Regions* service or the *Unsubscribe Object Class Attributes With Regions* service with an empty set of regions shall make no changes to the set of subscriptions with region of the class attribute.

   Any region specifications used to generate region realizations associated with a class attribute for subscription shall contain only dimensions that are a subset of the available dimensions of that class attribute. The resulting region realizations shall have the same property.

   The default region shall be used for subscription of a class attribute if and only if that class attribute has available dimensions, there is no other region realization used for subscription of that class attribute, and the joined federate is subscribed to that class attribute. Subsequently invoking the *Unsubscribe Object Class Attributes* service for the same object class and attribute shall cause the default region not to be used for subscription of that class attribute.

   A joined federate shall use a region for subscription of a class attribute to specify requirements for reflecting values of that class attribute's analogous instance attributes.

9.1.3.3 Interaction relationships

The following relationships, established through DDM services, shall pertain to interaction classes, parameters, and interactions:

a) A region realization other than the default region shall be used for sending an interaction if the joined federate has used the interaction and a region specification from which the region realization may be **derived** as arguments to an invocation of the *Send Interaction With Regions* service.

   Any region associated with a sent interaction shall contain only dimensions that are a subset of the available dimensions of the sent interaction. The default region shall be used for sending an interaction of such a class during an invocation of the *Send Interaction* service.

   A joined federate shall use a region for sending an interaction to assert properties of that interaction when the *Send Interaction With Regions* service is invoked.

b) A region realization other than the default region shall be used for subscription of an interaction class if the joined federate has used the interaction class and the region specification from which the region realization was derived as arguments in the *Subscribe Interaction Class With Regions* service. Subsequently invoking the *Unsubscribe Interaction Class With Regions* service for the same (interaction class, region) pair shall cause the region not to be used for subscription of that interaction class. Invoking the *Subscribe Interaction Class With Regions* service or the *Unsubscribe Interaction Class With Regions* service with an empty set of regions shall make no changes to the set of subscriptions with region of the interaction class.

   Any region associated with an interaction class for subscription shall contain only dimensions that are a subset of the available dimensions of that interaction class.

   The default region shall be used for subscription of that interaction class if and only if that interaction class has available dimensions, there is no other region realization used for subscription of that interaction class, and the joined federate is subscribed to that interaction class. Subsequently invoking the *Unsubscribe Interaction Class* service for the same interaction class shall cause the default region not to be used for subscription of that interaction class.

   A joined federate shall use a region for subscription of an interaction class to establish requirements for receiving interactions of that class.

A set of region realizations used for update of instance attributes or for sending interactions shall be called an **update region set**.

A set of region realizations used for subscription of either class attributes or interaction classes shall be called a **subscription region set**.

9.1.4 Calculating region overlaps

An update region set and a subscription region set shall overlap if and only if a region exists in each set so that the two regions overlap. Two regions overlap if and only if all ranges of dimensions that are contained in both regions overlap pairwise. If two regions do not have any dimensions in common, they shall not overlap. Two ranges in the same dimension $A = [a_{lower}, a_{upper})$ and $B = [b_{lower}, b_{upper})$ shall overlap if and only if either $a_{lower} = b_{lower}$ or ($a_{lower} < b_{upper}$ and $b_{lower} < a_{upper}$). The default region shall overlap with all nonempty region realizations.

The normalization of federation data to [0, a particular dimension's upper bound) for use with DDM services shall be left to the federation. The effects of DDM services shall be independent of the HLA time axis and HLA time management services.

Figure 18 depicts a region with two dimensions.

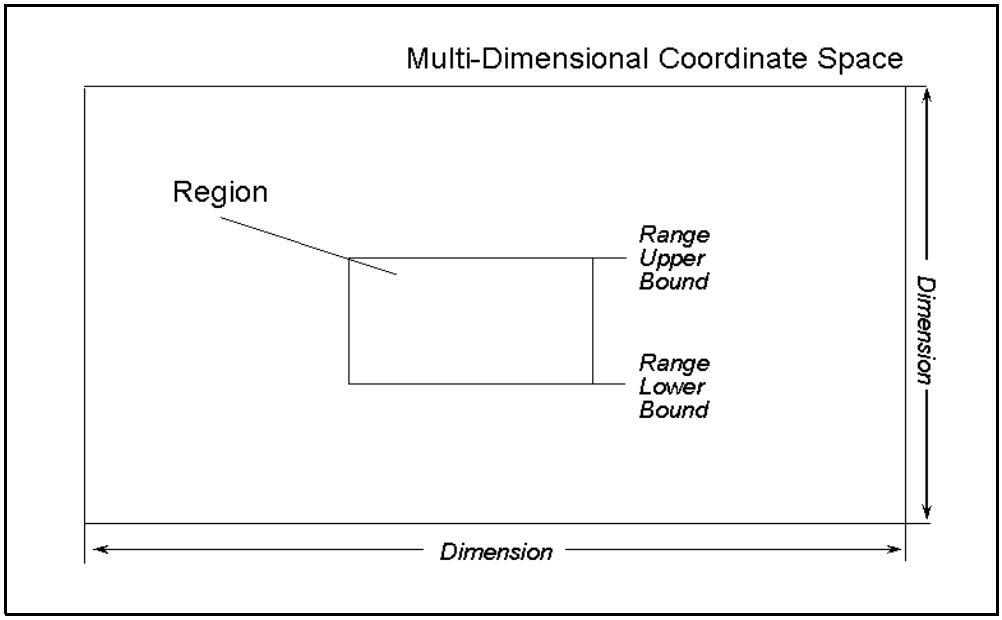

Figure 18—Region of two dimensions

9.1.5 Reinterpretation of selected DM services

Some DDM services can be used to perform similar functions to what is accomplished with DM services. When a joined federate uses DDM services, some of the DM definitions, constraints and services described in Clause 6 shall be extended to encompass the expanded interpretation of how DM services work when used in conjunction with DDM services by a joined federate, from the perspective of that joined federate.

A joined federate that is using DDM services shall interpret all uses of the following four DM services by any joined federate (including itself) in the federation execution:

— *Subscribe Object Class Attributes*
— *Unsubscribe Object Class Attributes*
— *Subscribe Interaction Class*
— *Unsubscribe Interaction Class*

as special cases of the following DDM services, respectively:

— *Subscribe Object Class Attributes With Regions*
— *Unsubscribe Object Class Attributes With Regions*
— *Subscribe Interaction Class With Regions*
— *Unsubscribe Interaction Class With Regions*

From the perspective of the joined federate that is using DDM services, each of the four DM services listed above shall be defined to be equivalent to the corresponding DDM service when invoked with a region argument of the default region.

In practice, because there shall be no way to refer to the default region, there shall be no way to substitute a DDM service for its corresponding DM service. Furthermore, a given joined federate may invoke both the DM services listed above and their corresponding DDM services using the same object class and class attribute designators or interaction class designators as arguments and there shall be no interaction between

the subscription effects that result from the DM service invocations and those that result from the DDM service invocations.

For a joined federate that is using DDM services, the following expanded definitions and constraints shall replace the correspondingly numbered DM definitions and constraints that appear in 5.1.2 and 5.1.3.

Expanded definitions and constraints replace corresponding items in 5.1.2 as follows:

a) A class attribute may be used as an argument to *Subscribe Object Class Attributes*, *Subscribe Object Class Attributes With Regions, Unsubscribe Object Class Attributes, Unsubscribe Object Class Attributes With Regions, Publish Object Class Attributes*, and *Unpublish Object Class Attributes* service invocations for a particular object class if and only if the attribute is an available attribute of that object class.

b) From a joined federate's perspective, the subscribed attributes of an object class shall be all of the class attributes that have been arguments to *Subscribe Object Class Attributes* service invocations by that joined federate for that object class and that have not subsequently been unsubscribed, either individually or by unsubscribing the whole object class. *Subscribe Object Class Attributes* and *Unsubscribe Object Class Attributes* service invocations for one object class shall have no effect on the subscribed attributes of any other object class.

   From a joined federate's perspective, the subscribed attributes of an object class with region shall be the class attributes that have been arguments to *Subscribe Object Class Attributes With Regions* service invocations by that joined federate for that object class and any region, assuming the joined federate did not subsequently invoke the *Unsubscribe Object Class Attributes With Regions* service for that (object class, region, class attribute) triple.

   Subscribe Object Class Attributes With Regions and *Unsubscribe Object Class Attributes With Regions* service invocations for one (object class, region, class attribute) triple shall have no effect on the subscribed attributes of any other (object class, region, class attribute) triples.

   Subscribe Object Class Attributes and *Unsubscribe Object Class Attributes* service invocations shall have no effect on any class attribute with region subscriptions that were established via the *Subscribe Object Class Attributes With Regions* service.

   Subscribe Object Class Attributes With Regions and *Unsubscribe Object Class Attributes With Regions* service invocations shall have no effect on any class attribute subscriptions that were established via the *Subscribe Object Class Attributes* service.

c) If a class attribute is a subscribed attribute of an object class, the joined federate shall be subscribed to that class attribute either actively or passively, but not both. If a class attribute is a subscribed attribute of an object class with region, the joined federate shall be subscribed to that class attribute at a given object class and region either actively or passively, but not both.

h) From a joined federate's perspective, an object class shall be subscribed if and only if
   1) The joined federate is subscribed to at least one attribute of the object class, or
   2) The joined federate is subscribed with region to at least one attribute of the object class.

j) Joined federates may invoke the *Register Object Instance* and the *Register Object Instance With Regions* services only with a published object class as an argument.

k) The registered class of an object instance shall be the object class that was an argument to either the *Register Object Instance* or the *Register Object Instance With Regions* service invocation for that object instance.

s) An update to an instance attribute by the joined federate that owns that instance attribute may be reflected only by other joined federates that are either
   1) Subscribed to the instance attribute's corresponding class attribute at the known class of the object instance that contains that instance attribute at the subscribing joined federate, or
   2) Using a subscription region set for the instance attribute's corresponding class attribute at the known class of the object instance at the subscribing joined federate, and this set overlaps the update region set of the specified instance attribute at the wall-clock time that the *Update*

Attribute Values service was invoked. An update shall be reflected at the subscribing joined federate at most once, regardless of another subscription to the class attribute using the *Subscribe Object Class Attributes* service.

Expanded definitions and constraints replace corresponding items in 5.1.3 as follows:

a) From a joined federate's perspective, an interaction class shall be subscribed if and only if
   1) It was an argument to a *Subscribe Interaction Class* service invocation by that joined federate that was not subsequently followed by an *Unsubscribe Interaction Class* service invocation for that interaction class, or
   2) There is at least one region where the interaction class and region were arguments to a *Subscribe Interaction Class With Regions* service invocation by that joined federate that was not subsequently followed by an *Unsubscribe Interaction Class With Regions* service invocation for that interaction class and region.

b) If an interaction class is subscribed, the joined federate shall be subscribed to that interaction class either actively or passively, but not both. If an interaction class is subscribed with region, the joined federate shall be subscribed to that interaction class with a given region either actively or passively, but not both.

d) Joined federates may invoke the *Send Interaction* and the *Send Interaction With Regions* services only with a published interaction class as an argument.

e) The sent class of an interaction shall be the interaction class that was an argument to the *Send Interaction* or the *Send Interaction With Regions* service invocation for that interaction.

g) The *Receive Interaction †* service shall be invoked at a joined federate only with a subscribed interaction class as an argument. An interaction shall be received at the joined federate at most once, regardless of another subscription to the interaction class using the *Subscribe Interaction Class* service.

j) Only the available parameters of an interaction class may be used in a *Send Interaction* and *Send Interaction With Regions* service invocation with that interaction class as an argument.

k) The sent parameters of an interaction shall be the parameters that were arguments to the *Send Interaction* or *Send Interaction With Regions* service invocation for that interaction.

9.1.6 Reinterpretation of selected object management services

Some DDM services can be used to perform similar functions to what is accomplished with object management services. When a joined federate uses DDM services, the RTI shall extend processing of three of the object management services described in Clause 6 to encompass the expanded interpretation of how object management services work when used in conjunction with DDM services by a joined federate, from the perspective of that joined federate.

A joined federate using DDM services shall interpret all uses of the following three object management services by any joined federate in the federation execution (including itself):

— *Register Object Instance*
— *Send Interaction*
— *Request Attribute Value Update*

as special cases of the following DDM services, respectively:

— *Register Object Instance With Regions*
— *Send Interaction With Regions*
— *Request Attribute Value Update With Regions*

From the perspective of the joined federate that is using DDM services, each of the three object management services listed above shall be defined to be equivalent to the corresponding DDM service when invoked with a region argument of the default region.

9.1.7 Convey Region Designator Sets Switch

Each federate's Convey Region Designator Sets Switch indicates whether the RTI shall provide the optional Set of Sent Region Designators argument with invocations of *Reflect Attribute Values †* and *Receive Interaction †* services at the joined federate.

The federation-wide initial setting for this switch shall come from the FDD supplied to the *Create Federation Execution* service invocation for the federation execution. This switch setting shall be modifiable during a federation execution via the HLAmanager.HLAfederation.HLAadjust.HLAsetSwitches MOM interaction.

Whenever this switch is enabled at a joined federate, the optional Set of Sent Region Designators argument shall be provided (as appropriate) with all *Reflect Attribute Values †* and *Receive Interaction †* service invocations at that joined federates. For the *Reflect Attribute Values †* service, if the specified instance attributes have available dimensions, the Set of Sent Region Designators argument shall contain the respective update region set. For the *Receive Interaction †* service, if the specified interaction has available dimensions, the Set of Sent Region Designators argument shall contain the respective update region set.

If the specified interaction or corresponding class attributes has available dimensions, this switch is enabled, and no region designator set is supplied, then the default region shall have been used on the sending side. In this case, the Convey Region set argument shall be present and empty.

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.

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 intact, 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 shall 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.

9.2 *Create Region* service

The *Create Region* service shall create a region that has the specified dimensions. The region may be used for either update or subscription.

9.2.1 Supplied arguments

a) Set of dimension designators.

9.2.2 Returned arguments

a) Region designator.

9.2.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified dimension designator set exists.
e) Federate save not in progress.
f) Federate restore not in progress.

9.2.4 Postconditions

a) A region has been created with the specified dimensions.

9.2.5 Exceptions

a) Invalid dimension designator.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

9.2.6 Reference state charts

a) None.

9.3 *Commit Region Modifications* service

The *Commit Region Modifications* service shall inform the RTI about changes to the ranges of the regions.

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.

9.3.1 Supplied arguments

a) Set of region designators.

9.3.2 Returned arguments

a) None.

9.3.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The regions exist.
e) The regions were created by the invoking joined federate using the *Create Region* service.
f) Federate save not in progress.

g) Federate restore not in progress.

9.3.4 Postconditions

a) The specified regions are modified.
b) All update region realizations derived from the region specifications are modified.
c) The RTI has been informed of the joined federate's requested change to all subscription region realizations derived from the region specifications.

9.3.5 Exceptions

a) The region was not created by the joined federate using the *Create Region* service.
b) Invalid region.
c) Federate save in progress.
d) Federate restore in progress.
e) The federate is not a federation execution member.
f) Not connected.
g) RTI internal error.

9.3.6 Reference state charts

a) None.

9.4 *Delete Region* service

The *Delete Region* service shall delete the specified region. A region shall not be deleted if there exist region realizations derived from that region that are used for update, subscription of a class attribute, or subscription of an interaction class.

9.4.1 Supplied arguments

a) Region designator.

9.4.2 Returned arguments

a) None.

9.4.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The region exists.
e) The region was created by the invoking joined federate using the *Create Region* service.
f) The region is not in use for update or subscription.
g) Federate save not in progress.
h) Federate restore not in progress.

9.4.4 Postconditions

a) The region no longer exists.

9.4.5 Exceptions

a) The region is in use for update or subscription.
b) The region was not created by the joined federate using the *Create Region* service.
c) Invalid region.
d) Federate save in progress.
e) Federate restore in progress.
f) The federate is not a federation execution member.
g) Not connected.
h) RTI internal error.

9.4.6 Reference state charts

a) None.

9.5 *Register Object Instance With Regions* service

The *Register Object Instance With Regions* service shall create a unique object instance designator and link it with an object instance of the supplied object class. All instance attributes of the object instance for which the corresponding class attributes are currently published by the registering joined federate shall be set as owned by the registering joined federate.

This service shall be used to create an object instance and simultaneously associate update regions with instance attributes of that object instance. This service shall be an atomic operation that can be used in place of the *Register Object Instance* service followed by the *Associate Regions For Updates* service when the atomicity is required. The instance attributes whose corresponding class attributes are currently published and that have available dimensions, but are not supplied in the service invocation, shall be associated with the default region.

If the region set paired with an instance attribute is empty and the corresponding class attribute of the instance attribute has available dimensions, that instance attribute shall be associated with the default region.

If a joined federate loses ownership of an instance attribute that it had associated with an update region, the association shall be lost. If the joined federate subsequently acquires ownership of that instance attribute, the original association shall not be restored.

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 a federation execution-wide unique name, and that name shall be coadunated with the object instance.

9.5.1 Supplied arguments

a) Object class designator.
b) Collection of attribute designator set and region designator set pairs.
c) Optional object instance name.

9.5.2 Returned arguments

a) Object instance handle.

9.5.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The object class is defined in the FDD.
e) The joined federate is publishing the object class.
f) The class attributes are available at the specified object class.
g) The joined federate is publishing the specified class attributes of the specified object class.
h) The regions exist.
i) The regions were created by the invoking joined federate using the *Create Region* service.
j) For each class attribute/region pair, the dimensions denoted by the region are a subset of the available dimensions of the specified class attributes in the FDD.
k) If the optional object instance name argument is supplied, that name is reserved and not already coadunated with another object instance.
l) 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.
m) Federate save not in progress.
n) Federate restore not in progress.

9.5.4 Postconditions

a) The returned object instance designator is associated with the object instance.
b) The joined federate owns the instance attributes that correspond to the class attributes that are published attributes at the specified object class.
c) The specified instance attributes are associated with the respective regions for future *Update Attribute Values* service invocations.
d) If the optional object instance name argument is supplied, that name is associated with the object instance.

9.5.5 Exceptions

a) The object instance name was already coadunated with another object instance (if optional object instance name argument is supplied).
b) The object instance name was not reserved (if optional object instance name argument is supplied).
c) The specified dimensions of the region are not a subset of the available dimensions of the specified class attributes in the FDD.
d) The region was not created by the joined federate using the *Create Region* service.
e) Invalid region.
f) The joined federate is not publishing the class attribute.
g) The joined federate is not publishing the object class.
h) The class attribute is not available at the known class of the object instance.
i) The object class is not defined in the FDD.
j) Federate save in progress.
k) Federate restore in progress.
l) The federate is not a federation execution member.
m) Not connected.
n) RTI internal error.

9.5.6 Reference state charts

a) Figure 10 {implicit reference}.
b) Figure 15 {implicit reference}.

c) Figure 16 {implicit reference}.

9.6 *Associate Regions For Updates* service

The *Associate Regions For Updates* service shall associate regions to be used for updates with instance attributes of a specific object instance.

A joined federate that has associated regions with an instance attribute for update should ensure that the regions adequately and correctly bound the portion of space defined by dimensions in which communication of updates to the instance attribute are relevant at the wall-clock time when an *Update Attribute Values* service is invoked.

The association shall be used by the *Update Attribute Values* service to route data to subscribers whose subscription region sets overlap the specified update region set. 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 and add the association of that instance attribute with the specified region. If an instance attribute is not associated with the default region, then invocation of the *Associate Regions For Updates* service shall add the specified regions to the set of associations of the specified instance attribute. No changes shall be made to the association set if the specified regions are already in the set of associations of the specified instance attributes.

If the region set paired with an instance attribute is empty, no associations shall be added for that instance attribute, not even the default region.

If a joined federate loses ownership of an instance attribute that it had associated with an update region, the association shall be lost. If the joined federate subsequently acquires ownership of that instance attribute, the original association shall not be restored.

9.6.1 Supplied arguments

a) Object instance designator.
b) Collection of attribute designator set and region designator set pairs.

9.6.2 Returned arguments

a) None.

9.6.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.
e) The joined federate knows about the object instance with the specified designator.
f) The class attributes are available at the specified object class.
g) The regions exist.
h) The regions were created by the invoking joined federate using the *Create Region* service.
i) For each instance attribute/region pair, the dimensions denoted by the region are a subset of the available dimensions of the corresponding class attributes in the FDD.
j) 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.
k) Federate save not in progress.

l) Federate restore not in progress.

9.6.4 Postconditions

a) The specified instance attributes are associated with the respective regions for future *Update Attribute Values* service invocations.

9.6.5 Exceptions

a) The dimensions denoted by the region are not a subset of the available dimensions of the specified class attributes in the FDD.
b) The region was not created by the joined federate using the *Create Region* service.
c) Invalid region.
d) The class attribute is not available.
e) The object instance is not known.
f) Federate save in progress.
g) Federate restore in progress.
h) The federate is not a federation execution member.
i) Not connected.
j) RTI internal error.

9.6.6 Reference state charts

a) None.

9.7 *Unassociate Regions For Updates* service

The *Unassociate Regions For Updates* service shall remove the association between the regions and the specified instance attributes. If one or more of the specified regions are 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.

Each of the instance attributes that are unassociated by the invocation of this service shall be associated with the default region if and only if they are not associated with any other update region.

If the region set paired with an instance attribute is empty, no associations shall be removed for that instance attribute.

9.7.1 Supplied arguments

a) Object instance designator.
b) Collection of attribute designator set and region designator set pairs.

9.7.2 Returned arguments

a) None.

9.7.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified designator exists.

e) The joined federate knows about the object instance with the specified designator.
f) The corresponding class attributes are available at the known class of the object instance at the invoking federate.
g) The regions exist.
h) The regions were created by the invoking joined federate using the *Create Region* service.
i) 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.
j) Federate save not in progress.
k) Federate restore not in progress.

9.7.4 Postconditions

a) The regions are no longer associated with the specified instance attributes.
b) All affected instance attributes that no longer have regions associated with them shall now be associated with the default region.

9.7.5 Exceptions

a) The region was not created by the joined federate using the *Create Region* service.
b) Invalid region.
c) The class attribute is not available.
d) The object instance is not known.
e) Federate save in progress.
f) Federate restore in progress.
g) The federate is not a federation execution member.
h) Not connected.
i) RTI internal error.

9.7.6 Reference state charts

a) None.

9.8 *Subscribe Object Class Attributes With Regions* service

The *Subscribe Object Class Attributes With Regions* service shall specify an object class for which the RTI shall begin notifying the joined federate of discovery of instantiated object instances when at least one of that object instance's instance attributes is in scope. This service and subsequent related RTI operations shall behave analogously to the *Subscribe Object Class Attributes* service as described in 5.6 and its subsequent related RTI operations. This service shall provide additional functionality in that the overlap of the relevant subscription and update region sets affects the subsequent RTI operations, as described in 9.1.5.

This service shall add the specified regions to the set of subscriptions of the specified class attributes. No changes shall be made to the subscription set if the specified regions are already in the set of subscriptions of the specified class attributes.

Invocations of the *Subscribe Object Class Attributes With Regions* service shall have no affect on any object class or class attribute subscriptions that were established via the *Subscribe Object Class Attributes* service. Subscriptions that are established via the *Subscribe Object Class Attributes With Regions* service shall not be affected by invocations of either the *Subscribe Object Class Attributes* service or the *Unsubscribe Object Class Attributes* service.

If the region set paired with a class attribute is empty, no subscriptions shall be added for that class attribute, not even the default region. Use of an empty set is not equivalent to invocation of the *Subscribe Object Class Attributes* service.

If the optional passive subscription indicator indicates that this is a passive subscription, then the following apply:

a) The invocation of this service shall not cause the *Start Registration For Object Class †* service or the *Turn Updates On For Object Instance †* service to be invoked at any other joined federate, and
b) If a specified region was used in a previous subscription that was active rather than passive, this invocation shall change the previous subscription to passive. Invocation of this service may cause the *Stop Registration for Object Class †* service or the *Turn Updates Off For Object Instance †* service to be invoked at one or more other joined federates.

If the optional passive subscription indicator is not present or indicates that this is an active subscription, the invocation of this service may cause the *Start Registration For Object Class †* service or the *Turn Updates On For Object Instance †* service to be invoked at one or more other joined federates.

The use of the optional passive subscription indicator shall act on the triple basis, which is as follows:

c) 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 actively and one passively), 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. In other words, the active/passive characteristic is a property of a subscription to a class attribute at a given object class with a given region.
d) Each (object class, class attribute, region) triple specified in a given invocation of the *Subscribe Object Class Attributes With Regions* service shall 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 shall take on the effect of the optional active/passive subscription indicator supplied (or not supplied) with the current service invocation.
e) Invoking the *Subscribe Object Class Attributes With Regions* service with an (object class, class attribute, region set) triple where 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.

If the optional maximum update rate is specified, the attributes shall be considered to be subscribed with the corresponding update rate. In case no update rate is specified, the attributes shall be considered to be subscribed with the default update rate (no update rate reduction shall take place).

9.8.1 Supplied arguments

a) Object class designator.
b) Collection of attribute designator set and region designator set pairs.
c) Optional passive subscription indicator.
d) Optional update rate designator.

9.8.2 Returned arguments

a) None.

9.8.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The object class is defined in the FDD.
e) The class attributes are available at the specified object class.
f) The regions exist.
g) The regions were created by the invoking joined federate using the *Create Region* service.
h) For each class attribute/region pair, the dimensions denoted by the region are a subset of the available dimensions of the specified class attributes in the FDD.
i) 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.
j) Federate save not in progress.
k) Federate restore not in progress.

9.8.4 Postconditions

a) The RTI has been informed of the joined federate's requested subscription.

9.8.5 Exceptions

a) The specified dimensions of the region are not a subset of the available dimensions of the specified class attributes in the FDD.
b) The region was not created by the joined federate using the *Create Region* service.
c) Invalid region.
d) The class attribute is not available at the specified object class.
e) The object class is not defined in the FDD.
f) Invalid update rate designator
g) Federate save in progress.
h) Federate restore in progress.
i) The federate is not a federation execution member.
j) Not connected.
k) RTI internal error.

9.8.6 Reference state charts

a) Figure 11 {implicit reference}.
b) Figure 12 {implicit reference}.

9.9 *Unsubscribe Object Class Attributes With Regions* service

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. The unsubscribe shall be confined to only the specified class attribute subscriptions using the specified region. No changes shall be made to the subscription set of the

specified class attributes if the specified regions are not in the set of subscriptions of the specified class attributes.

If the region set paired with a class attribute is empty, no subscriptions shall be removed for that class attribute. Use of an empty set is not equivalent to invocation of the *Unsubscribe Object Class Attributes* service.

Invocation of this service, subject to other conditions, may cause an implicit out-of-scope situation for affected instance attributes, i.e., the invoking joined federate will NOT be notified via an invocation of the *Attributes Out Of Scope †* service when the instance attribute goes out of scope.

9.9.1 Supplied arguments

a) Object class designator.
b) Collection of attribute designator set and region designator set pairs.

9.9.2 Returned arguments

a) None.

9.9.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The object class is defined in the FDD.
e) The class attributes are available at the specified object class.
f) The regions exist.
g) The regions were created by the joined federate using the *Create Region* service.
h) 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.
i) Federate save not in progress.
j) Federate restore not in progress.

9.9.4 Postconditions

a) The joined federate is not subscribed to the specified class attributes with the specified regions.
b) May cause an implicit out-of-scope situation for affected instance attributes at invoking joined federate.

9.9.5 Exceptions

a) The region was not created by the invoking joined federate using the *Create Region* service.
b) Invalid region.
c) The class attribute is not available at the specified object class.
d) The object class is not defined in the FDD.
e) Federate save in progress.
f) Federate restore in progress.
g) The federate is not a federation execution member.
h) Not connected.
i) RTI internal error.

9.9.6 Reference state charts

a) Figure 11 {implicit reference}.
b) Figure 12 {implicit reference}.

9.10 *Subscribe Interaction Class With Regions* service

The *Subscribe Interaction Class With Regions* service shall specify the class of interactions that shall be delivered to the joined federate, taking the region into account. This service and subsequent related RTI operations shall behave analogously to the *Subscribe Interaction Class* service as described in 5.8. This service shall provide additional functionality in that the overlap of the region set used for subscription of the interaction and the region set used for sending the interaction affects the subsequent RTI operations, as described in 9.1.5.

This service shall add the specified regions to the set of regions associated with the specified interaction class subscription. No changes shall be made to the subscription set if the specified regions are already in the set of subscriptions of the specified interaction class.

Invocations of the *Subscribe Interaction Class With Regions* service shall have no affect on any interaction class subscriptions that were established via the *Subscribe Interaction Class* service. Subscriptions that are established via the *Subscribe Interaction Class With Regions* service shall not be affected by invocations of either the *Subscribe Interaction Class* service or the *Unsubscribe Interaction Class* service.

If the region set provided is empty, no subscription to the interaction class shall be made, not even the default region. Use of an empty set is not equivalent to invocation of the *Subscribe Interaction Class* service.

If the optional passive subscription indicator indicates that this is a passive subscription, than the following apply:

a) The invocation of this service shall not cause the *Turn Interactions On †* service to be invoked at any other joined federate, and
b) If a specified region was used in a previous subscription that was active rather than passive, this service invocation shall change the previous subscription to passive. Invocation of this service may cause the *Turn Interactions Off †* service to be invoked at one or more other joined federates.

If the optional passive subscription indicator is not present or indicates that this is an active subscription, the invocation of this service may cause the *Turn Interactions On †* service to be invoked at one or more other joined federates.

The use of the optional passive subscription indicator shall act as follows:

c) Each subscribed interaction class with regions shall be 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 actively and one passively), 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 shall 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 shall take on the effect of the optional passive/active subscription indicator supplied (or not supplied) with the service invocation.

d) Invoking the *Subscribe Interaction Class With Regions* service with an (interaction class, region set) pair where 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.

If a joined federate has the Service Reporting Switch Enabled, that joined federate shall not be able to subscribe to the HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation interaction class.

9.10.1 Supplied arguments

a) Interaction class designator.
b) Set of region designators.
c) Optional passive subscription indicator.

9.10.2 Returned arguments

a) None.

9.10.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The interaction class is defined in the FDD.
e) The regions exist.
f) The regions were created by the invoking joined federate using the *Create Region* service.
g) The specified dimensions of the regions are a subset of the available dimensions of the specified interaction class in the FDD.
h) If the specified interaction class designator is the MOM interaction class HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation, the federate's service invocations are not being reported via MOM interactions.
i) 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.
j) Federate save not in progress.
k) Federate restore not in progress.

9.10.4 Postconditions

a) The RTI has been informed of the joined federate's requested subscription.

9.10.5 Exceptions

a) Federate service invocations are being reported via MOM interactions.
b) The specified dimensions of the region are not a subset of the available dimensions of the specified interaction class in the FDD.
c) The region was not created by the joined federate using the *Create Region* service.
d) Invalid region.
e) The interaction class is not defined in the FDD.
f) Federate save in progress.
g) Federate restore in progress.
h) The federate is not a federation execution member.

i) Not connected.
j) RTI internal error.

9.10.6 Reference state charts

a) Figure 13 {implicit reference}.

9.11 *Unsubscribe Interaction Class With Regions* service

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. No changes shall be made to the subscription set of the specified interaction class if the specified regions are not in the set of subscriptions of the specified interaction class.

If the region set provided is empty, no subscription to the interaction class shall be removed. Use of an empty set is not equivalent to invocation of the *Unsubscribe Interaction Class* service.

9.11.1 Supplied arguments

a) Interaction class designator.
b) Set of region designators.

9.11.2 Returned arguments

a) None.

9.11.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The interaction class is defined in the FDD.
e) The regions exist.
f) The regions were created by the invoking joined federate using the *Create Region* service.
g) 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.
h) Federate save not in progress.
i) Federate restore not in progress.

9.11.4 Postconditions

a) The joined federate is not subscribed to the specified interaction class with the specified regions.

9.11.5 Exceptions

a) The region was not created by the joined federate using the *Create Region* service.
b) Invalid region.
c) The interaction class is not defined in the FDD.
d) Federate save in progress.
e) Federate restore in progress.
f) The federate is not a federation execution member.

g) Not connected.
h) RTI internal error.

9.11.6 Reference state charts

a) Figure 13 {implicit reference}.

9.12 *Send Interaction With Regions* service

The *Send Interaction With Regions* service shall send an interaction into the federation. Only parameters that are available at the specified 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. The regions shall be used to limit the scope of potential receivers of the interaction.

If preferred order type (see 8.1.1) of the interaction is TSO, the joined federate invoking this service is time-regulating, and the timestamp argument is present, this service shall return a federation-unique message retraction designator. Otherwise, no message retraction designator shall be returned. A timestamp value should be provided if the preferred order type of the interaction is TSO and the joined federate invoking this service is time-regulating.

Following the invocation of this service, if, at any wall-clock time when the RTI has not yet invoked a corresponding *Receive Interaction †* service at a potentially receiving joined federate, that joined federate is either

a) Unsubscribed to the specified interaction class, or
b) Using a subscription region set for the specified interaction class and this region set does not overlap the update region set used for sending the interaction,

then the RTI shall not invoke the corresponding *Receive Interaction †* service at that joined federate.

If, at any wall-clock time between the invocation of this service for a particular interaction class and what would be the invocation of the corresponding *Receive Interaction †* service at a potential receiving joined federate, that joined federate is either

c) Subscribed to the specified interaction class, or
d) Using a subscription region set for the specified interaction class, and this region set overlaps the update region set used for sending the interaction

then the RTI may invoke the corresponding *Receive Interaction †* service at that joined federate, subject to the preconditions of the *Receive Interaction †* service and the rules of time management.

If the region set provided is empty, the interaction shall not be sent. Use of an empty set is not equivalent to invocation of the *Send Interaction* service.

9.12.1 Supplied arguments

a) Interaction class designator.
b) Constrained set of parameter designator and value pairs.
c) Set of region designators.
d) User-supplied tag.
e) Optional timestamp.

9.12.2 Returned arguments

a) Optional message retraction designator.

9.12.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The interaction class is defined in the FDD.
e) The joined federate is publishing the interaction class.
f) The interaction parameters are available.
g) The regions exist.
h) The regions were created by the invoking joined federate using the *Create Region* service.
i) The specified dimensions of the regions are a subset of the available dimensions of the specified interaction class in the FDD.
j) The timestamp value shall be in accordance with the constraints stated in Clause 8 (if optional timestamp argument is supplied).
k) Only parameters that are available at the specified 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.
l) Federate save not in progress.
m) Federate restore not in progress.

9.12.4 Postconditions

a) The RTI has received the interaction.

9.12.5 Exceptions

a) The timestamp is invalid (if optional timestamp argument is supplied).
b) The specified dimensions of the region are not a subset of the available dimensions of the specified interaction class in the FDD.
c) The region was not created by the joined federate using the *Create Region* service.
d) Invalid region.
e) The joined federate is not publishing the specified interaction class.
f) The interaction parameter is not available at the specified interaction class.
g) The interaction class is not defined in the FDD.
h) Federate save in progress.
i) Federate restore in progress.
j) The federate is not a federation execution member.
k) Not connected.
l) RTI internal error.

9.12.6 Reference state charts

a) None.

9.13 *Request Attribute Value Update With Regions* service

The *Request Attribute Value Update With Regions* service shall be used to stimulate the update of values of specified attributes. The RTI shall solicit the values of the specified instance attributes, for all object instances registered at that class and all subclasses of that class, from their owners using the *Provide*

Attribute Value Update † service. The resulting *Provide Attribute Value Update †* service invocations issued by the RTI shall be consistent with the region sets provided to this service. An invocation shall be consistent with the provided region sets if the instance attributes in an updating joined federate are associated with an update region set that overlaps with the corresponding region set.

The timestamp of any resulting *Reflect Attribute Values †* service invocations shall be determined by the updating joined federate.

If the federate invoking this service also owns some of the instance attributes whose values are being requested, it shall not have the *Provide Attribute Value Update†* service invoked on it by the RTI. The request is taken as an implicit invocation of the provide service at the requesting federate for all qualified instance attributes that it owns.

This service does not allow the use of object instances. If a federate wants to request the update of instance attributes of a single object instance, use of the *Request Attribute Value Update* service is sufficient, and the use of DDM provides no additional benefit.

If the region set paired with a class attribute is empty, no request for update shall be issued for the class attribute, not even with the default region. Use of an empty set is not equivalent to invocation of the *Request Attribute Value Update* service.

The user-supplied tag argument shall be provided with all corresponding *Provide Attribute Value Update †* service invocations.

9.13.1 Supplied arguments

a) Object class designator.
b) Collection of attribute designator set and region designator set pairs.
c) User-supplied tag.

9.13.2 Returned arguments

a) None.

9.13.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The object class is defined in the FDD.
e) The class attributes are available at the specified object class.
f) The regions exist.
g) The regions were created by the invoking joined federate using the *Create Region* service.
h) The specified dimensions of the regions are a subset of the available dimensions of the specified class attributes in the FDD.
i) 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.
j) Federate save not in progress.
k) Federate restore not in progress.

9.13.4 Postconditions

a) The request for the updated attribute values has been received by the RTI.

9.13.5 Exceptions

a) The specified dimensions of the region are not a subset of the available dimensions of the specified class attributes in the FDD.
b) The region was not created by the joined federate using the *Create Region* service.
c) Invalid region.
d) The class attribute is not available.
e) The object class is not defined in the FDD
f) Federate save in progress.
g) Federate restore in progress.
h) The joined federate is not a federation execution member.
i) Not connected.
j) RTI internal error.

9.13.6 Reference state charts

a) None.

10. Support services

10.1 Overview

This clause describes miscellaneous services utilized by joined federates for performing such actions as the following:

— Performing name-to-handle and handle-to-name transformations
— Setting advisory switches
— Manipulating regions
— Getting update rates values.

10.1.1 Names

All class name arguments shall be completely specified, including all superclass names, with the exception of the HLA root classes. The inclusion of HLAobjectRoot or HLAinteractionRoot shall be optional.

Each name in an FDD (object classes, interactions, attributes, parameters, dimensions, transportation types, order types) shall have a unique, and unpredictable, federation execution-wide handle.

All names shall be case sensitive.

10.1.2 Advisory switches

The settings of the following advisory switches may be modified by services in Clause 10:

— Object Class Relevance Advisory Switch
— Attribute Relevance Advisory Switch
— Attribute Scope Advisory Switch
— Interaction Relevance Advisory Switch

The federation-wide initial setting of each advisory switch shall be as indicated in the FDD supplied to the *Create Federation Execution* service invocation for the federation execution. Subsequent invocation of the advisory switch services in this clause shall affect a switch setting for the invoking federate only.

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. The disabling of an advisory switch directs the RTI to cease informing the joined federate of changes in the conditions covered by that advisory. When an advisory switch is disabled, all relevant notifications that would have been delivered to the joined federate shall be lost, i.e., the RTI shall not save the relevant actions and send them to the joined federate when a switch is subsequently enabled.

There is, however, one case in which the RTI shall not inform a joined federate when specific conditions covered by an advisory change, even if that advisory switch is enabled. Specifically, even if the Attribute Scope Advisory Switch is enabled, there are some actions a joined federate could take that would cause an instance attribute to go out of scope with respect to that joined federate. Since these changes are a result of the joined federate's own action and the joined federate knows that the action will cause the instance attribute to go out of scope, no out-of-scope advisory notification shall be given the joined federate. The actions that could cause an instance attribute to go out of scope for the acting joined federate are as follows:

— The joined federate acquires ownership of the instance attribute.
— The joined federate unsubscribes to the instance attribute's corresponding class attribute, and it is not subscribed to the instance attribute's corresponding class attribute with region.

— The joined federate unsubscribes with region to the instance attribute's corresponding class attribute so that it is no longer subscribed to the class attribute (with or without region).

These cases are referred to as implicit out-of-scope situations because no out-of-scope advisory notification is generated.

Note that if a joined federate unsubscribes with or without region to an instance attribute's corresponding class attribute, but is still subscribed to that class attribute with some other region, it may or may not have caused the instance attribute to go out of scope (assuming it was previously in scope). Since the joined federate does not know if its removal has caused overlap to be lost with the update region set, it must in this case rely on the RTI to inform it of whether the instance attribute has gone out of scope. Only if the joined federate completely removes all subscriptions (DM and DDM based subscriptions) can it know with certainty that it is causing an implicit out-of-scope situation.

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 discerned by that federate, is as follows:

a) 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.
b) 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 13).
c) 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 15).
d) 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 15).

10.2 *Get Automatic Resign Directive* service

The *Get Automatic Resign Directive* service shall return the current Automatic Resign Directive for the federation execution.

10.2.1 Supplied arguments

a) None.

10.2.2 Returned arguments

a) Automatic resign directive.

10.2.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.

10.2.4 Post conditions

a) The federate has the Automatic Resign Directive.

10.2.5 Exceptions

a) The federate is not a federation execution member.
b) Not connected.
c) RTI internal error.

10.2.6 Reference state charts

a) None.

10.3 *Set Automatic Resign Directive* service

The *Set Automatic Resign Directive* service shall set the Automatic Resign Directive for the federation execution.

10.3.1 Supplied arguments

a) Automatic resign directive.

10.3.2 Returned arguments

a) None.

10.3.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.

10.3.4 Post conditions

a) The Automatic Resign Directive for the federation execution is set to the specified value.

10.3.5 Exceptions

a) Invalid resign action.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

10.3.6 Reference state charts

a) None.

10.4 *Get Federate Handle* service

The *Get Federate Handle* service shall return the handle of the joined federate with the supplied name.

10.4.1 Supplied arguments

a) Federate name.

10.4.2 Returned arguments

a) Federate handle.

10.4.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The federate with the supplied name exists.

10.4.4 Postconditions

a) The joined federate has the requested federate handle.

10.4.5 Exceptions

a) The supplied federate name is not understood.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

10.4.6 Reference state charts

a) None.

10.5 *Get Federate Name* service

The *Get Federate Name* service shall return the name of the federate with the supplied handle.

10.5.1 Supplied arguments

a) Federate handle.

10.5.2 Returned arguments

a) Federate name.

10.5.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The federate with the specified handle exists.

10.5.4 Postconditions

a) The joined federate has the requested federate name.

10.5.5 Exceptions

a) Invalid federate handle.
b) The supplied federate handle is not understood.

c) The federate is not a federation execution member.
d) Not connected.
e) RTI internal error.

10.5.6 Reference state charts

a) None.

10.6 *Get Object Class Handle* service

The *Get Object Class Handle* service shall return the object class handle associated with the supplied object class name.

10.6.1 Supplied arguments

a) Object class name.

10.6.2 Returned arguments

a) Object class handle.

10.6.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified object class is defined in the FDD.

10.6.4 Postconditions

a) The joined federate has the requested object class handle.

10.6.5 Exceptions

a) The object class is not defined in the FDD.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

10.6.6 Reference state charts

a) None.

10.7 *Get Object Class Name* service

The *Get Object Class Name* service shall return the object class name associated with the supplied object class handle.

10.7.1 Supplied arguments

a) Object class handle.

10.7.2 Returned arguments

a) Object class name.

10.7.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to the federation execution.
d) The specified object class handle exists.

10.7.4 Postconditions

a) The joined federate has the requested object class name.

10.7.5 Exceptions

a) Invalid object class handle.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

10.7.6 Reference state charts

a) None.

10.8 *Get Known Object Class Handle* service

The *Get Known Object Class Handle* service shall return the known object class of the supplied object instance.

10.8.1 Supplied arguments

a) Object instance handle.

10.8.2 Returned arguments

a) Object class handle.

10.8.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) An object instance with the specified handle exists.
e) The joined federate knows about the object instance with the specified handle.

10.8.4 Postconditions

a) The joined federate has the known object class handle of the specified object instance.

10.8.5 Exceptions

a) Invalid object instance handle.
b) The object instance is not known.
c) The federate is not a federation execution member.
d) Not connected.
e) RTI internal error.

10.8.6 Reference state charts

a) None.

10.9 *Get Object Instance Handle* service

The *Get Object Instance Handle* service shall return the handle of the object instance with the supplied name.

10.9.1 Supplied arguments

a) Object instance name.

10.9.2 Returned arguments

a) Object instance handle.

10.9.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The object instance with the specified name exists.
e) The joined federate knows about the object instance with the specified name.

10.9.4 Postconditions

a) The joined federate has the requested object instance handle.

10.9.5 Exceptions

a) The object instance is not known.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

10.9.6 Reference state charts

a) None.

10.10 *Get Object Instance Name* service

The *Get Object Instance Name* service shall return the name of the object instance with the supplied handle.

10.10.1 Supplied arguments

a) Object instance handle.

10.10.2 Returned arguments

a) Object instance name.

10.10.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The object instance with the specified handle exists.
e) The joined federate knows about the object instance with the specified handle.

10.10.4 Postconditions

a) The joined federate has the requested object instance name.

10.10.5 Exceptions

a) Invalid object instance handle.
b) The object instance is not known.
c) The federate is not a federation execution member.
d) Not connected.
e) RTI internal error.

10.10.6 Reference state charts

a) None.

10.11 *Get Attribute Handle* service

The *Get Attribute Handle* service shall return the class attribute handle associated with the supplied class attribute name and object class.

10.11.1 Supplied arguments

a) Object class handle.
b) Class attribute name.

10.11.2 Returned arguments

a) Class attribute handle.

10.11.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.

c) The federate is joined to that federation execution.
d) The specified object class handle exists.
e) The specified class attribute is an available attribute of the specified object class.

10.11.4 Postconditions

a) The joined federate has the requested class attribute handle.

10.11.5 Exceptions

a) The object class attribute is not an available attribute of the object class.
b) Invalid object class handle.
c) The federate is not a federation execution member.
d) Not connected.
e) RTI internal error.

10.11.6 Reference state charts

a) None.

10.12 *Get Attribute Name* service

The *Get Attribute Name* service shall return the class attribute name associated with the supplied class attribute handle and object class.

10.12.1 Supplied arguments

a) Object class handle.
b) Class attribute handle.

10.12.2 Returned arguments

a) Class attribute name.

10.12.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified object class handle exists.
e) The specified class attribute handle exists.
f) The specified class attribute is an available attribute of the specified object class.

10.12.4 Postconditions

a) The joined federate has the requested class attribute name.

10.12.5 Exceptions

a) The object class attribute is not an available attribute of the object class.
b) Invalid class attribute handle.
c) Invalid object class handle.
d) The federate is not a federation execution member.

e) Not connected.
f) RTI internal error.

10.12.6 Reference state charts

a) None.

10.13 *Get Update Rate Value* service

The *Get Update Rate Value* service shall return the maximum update rate value for the specified update rate.

10.13.1 Supplied arguments

a) Update rate name.

10.13.2 Returned arguments

a) Maximum update rate value.

10.13.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The update rate name exists in the FDD.

10.13.4 Post conditions

a) The joined federate has the maximum update rate value.

10.13.5 Exceptions

a) Invalid update rate name.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

10.13.6 Reference state charts

a) None.

10.14 *Get Update Rate Value For Attribute* service

The *Get Update Rate Value For Attribute* service shall return the maximum update rate value for the specified attribute of the specified object instance.

10.14.1 Supplied arguments

a) Object instance handle.
b) Attribute handle.

10.14.2 Returned arguments

a) Maximum update rate value.

10.14.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The object instance exists.
e) The attribute handle is valid for the object instance.

10.14.4 Post conditions

a) The joined federate has the maximum update rate value for the object instance attribute.

10.14.5 Exceptions

a) Invalid object instance handle.
b) The object instance is not known.
c) Invalid attribute handle.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

10.14.6 Reference state charts

a) None.

10.15 *Get Interaction Class Handle* service

The *Get Interaction Class Handle* service shall return the interaction class handle associated with the supplied interaction class name.

10.15.1 Supplied arguments

a) Interaction class name.

10.15.2 Returned arguments

a) Interaction class handle.

10.15.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified interaction class is defined in the FDD.

10.15.4 Postconditions

a) The joined federate has the requested interaction class handle.

10.15.5 Exceptions

a) The interaction class is not defined in the FDD.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

10.15.6 Reference state charts

a) None.

10.16 *Get Interaction Class Name* service

The *Get Interaction Class Name* service shall return the interaction class name associated with the supplied interaction class handle.

10.16.1 Supplied arguments

a) Interaction class handle.

10.16.2 Returned arguments

a) Interaction class name.

10.16.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified interaction class handle exists.

10.16.4 Postconditions

a) The joined federate has the requested interaction class name.

10.16.5 Exceptions

a) Invalid interaction class handle.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

10.16.6 Reference state charts

a) None.

10.17 *Get Parameter Handle* service

The *Get Parameter Handle* service shall return the interaction parameter handle associated with the supplied parameter name and interaction class.

10.17.1 Supplied arguments

a) Interaction class handle.
b) Parameter name.

10.17.2 Returned arguments

a) Parameter handle.

10.17.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified interaction class handle exists.
e) The specified parameter is an available parameter of the specified interaction class.

10.17.4 Postconditions

a) The joined federate has the requested parameter handle.

10.17.5 Exceptions

a) The parameter is not an available parameter of the interaction class.
b) Invalid interaction class handle.
c) The federate is not a federation execution member.
d) Not connected.
e) RTI internal error.

10.17.6 Reference state charts

a) None.

10.18 *Get Parameter Name* service

The *Get Parameter Name* service shall return the interaction parameter name associated with the supplied parameter handle and interaction class.

10.18.1 Supplied arguments

a) Interaction class handle.
b) Parameter handle.

10.18.2 Returned arguments

a) Parameter name.

10.18.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified interaction class handle exists.
e) The specified parameter handle exists.
f) The specified parameter is an available parameter of the specified interaction class.

10.18.4 Postconditions

a) The joined federate has the requested parameter name.

10.18.5 Exceptions

a) The parameter is not an available parameter of the interaction class.
b) Invalid parameter handle.
c) Invalid interaction class handle.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

10.18.6 Reference state charts

a) None.

10.19 *Get Order Type* service

The *Get Order Type* service shall return the order type associated with the supplied order name, as defined in 8.1.8.

10.19.1 Supplied arguments

a) Order name.

10.19.2 Returned arguments

a) Order type.

10.19.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The order name is valid (see 8.1.11).

10.19.4 Postconditions

a) The joined federate has the requested order type.

10.19.5 Exceptions

a) Invalid order name.

b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

10.19.6 Reference state charts

a) None.

10.20 *Get Order Name* service

The *Get Order Nam*e service shall return the order name, as defined in 8.1.11, associated with the supplied order type.

10.20.1 Supplied arguments

a) Order type.

10.20.2 Returned arguments

a) Order name.

10.20.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The order type is valid (8.1.11).

10.20.4 Postconditions

a) The joined federate has the requested order name.

10.20.5 Exceptions

a) Invalid order type.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

10.20.6 Reference state charts

a) None.

10.21 *Get Transportation Type Handle* service

The *Get Transportation Type Handle* service shall return the transportation type handle associated with the supplied transportation name.

10.21.1 Supplied arguments

a) Transportation type name.

10.21.2 Returned arguments

a) Transportation type handle.

10.21.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified transportation type is defined in the FDD.

10.21.4 Postconditions

a) The joined federate has the requested transportation type handle.

10.21.5 Exceptions

a) The transportation type is not defined in the FDD.
b) The federate is not a federation execution member.
c) Not connected
d) RTI internal error.

10.21.6 Reference state charts

a) None.

10.22 *Get Transportation Type Name* service

The *Get Transportation Type Name* service shall return the transportation type name associated with the supplied transportation type handle.

10.22.1 Supplied arguments

a) Transportation type handle.

10.22.2 Returned arguments

a) Transportation type name.

10.22.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified transportation type handle exists.

10.22.4 Postconditions

a) The joined federate has the requested transportation type name.

10.22.5 Exceptions

a) Invalid transportation type handle.

b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

10.22.6 Reference state charts

a) None.

10.23 *Get Available Dimensions For Class Attribute* service

The *Get Available Dimensions For Class Attribute* service shall return the available dimensions of the supplied class attribute and object class. If the supplied class attribute does not have available dimensions, an empty set shall be returned.

10.23.1 Supplied arguments

a) Object class handle.
b) Class attribute handle.

10.23.2 Returned arguments

a) A set of dimension handles.

10.23.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified object class handle exists.
e) The specified class attribute handle exists.
f) The specified class attribute is an available attribute of the specified object class.

10.23.4 Postconditions

a) The joined federate has the dimension handle set.

10.23.5 Exceptions

a) The object class attribute is not an available attribute of the object class.
b) Invalid class attribute handle.
c) Invalid object class handle.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

10.23.6 Reference state charts

a) None.

10.24 *Get Available Dimensions For Interaction Class* service

The *Get Available Dimensions For Interaction Class* service shall return the available dimensions of the supplied interaction class. If the supplied interaction class does not have available dimensions, an empty set shall be returned.

10.24.1 Supplied arguments

a) Interaction class handle.

10.24.2 Returned arguments

a) A set of dimension handles.

10.24.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified interaction class handle exists.

10.24.4 Postconditions

a) The joined federate has the requested dimension handle set.

10.24.5 Exceptions

a) Invalid interaction class handle.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

10.24.6 Reference state charts

a) None.

10.25 *Get Dimension Handle* service

The *Get Dimension Handle* service shall return the dimension handle associated with the supplied dimension name.

10.25.1 Supplied arguments

a) Dimension name.

10.25.2 Returned arguments

a) Dimension handle.

10.25.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.

c) The federate is joined to that federation execution.
d) The specified dimension is defined in the FDD.

10.25.4 Postconditions

a) The joined federate has the requested dimension handle.

10.25.5 Exceptions

a) The dimension is not defined in the FDD.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

10.25.6 Reference state charts

a) None.

10.26 *Get Dimension Name* service

The *Get Dimension Name* service shall return the dimension name associated with the supplied dimension handle.

10.26.1 Supplied arguments

a) Dimension handle.

10.26.2 Returned arguments

a) Dimension name.

10.26.3 Preconditions

b) The federate is connected to the RTI.
a) The federation execution exists.
b) The federate is joined to that federation execution.
c) The specified dimension handle exists.

10.26.4 Postconditions

a) The joined federate has the requested dimension name.

10.26.5 Exceptions

a) Invalid dimension handle.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

10.26.6 Reference state charts

a) None.

10.27 *Get Dimension Upper Bound* service

The *Get Dimension Upper Bound* service shall return the positive integer that is the upper bound for the specified dimension, as specified in the FDD.

10.27.1 Supplied arguments

a) Dimension handle.

10.27.2 Returned arguments

a) Dimension upper bound.

10.27.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The specified dimension handle exists.

10.27.4 Postconditions

a) The joined federate has the requested dimension upper bound.

10.27.5 Exceptions

a) Invalid dimension handle.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

10.27.6 Reference state charts

a) None.

10.28 *Get Dimension Handle Set* service

The *Get Dimension Handle Set* service shall return the dimensions of the specified region.

10.28.1 Supplied arguments

a) Region handle.

10.28.2 Returned arguments

a) A set of dimensions.

10.28.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The region exists.

e) The region was either created by the invoking joined federate using the *Create Region* service or 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.

10.28.4 Postconditions

a) The joined federate has the dimensions of the region.

10.28.5 Exceptions

a) Invalid region handle.
b) Invalid region.
c) Federate save in progress.
d) Federate restore in progress.
e) The federate is not a federation execution member.
f) Not connected.
g) RTI internal error.

10.28.6 Reference state charts

a) None.

10.29 *Get Range Bounds* service

The *Get Range Bounds* service shall return the current lower and upper bounds of the range for the specified dimension in the specified region.

10.29.1 Supplied arguments

a) Region handle.
b) Dimension handle.

10.29.2 Returned arguments

a) Range lower bound.
b) Range upper bound.

10.29.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The region exists.
e) The region contains the specified dimension.
f) Either the region was 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.

g) All supplied region designators are designators either of 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.
h) Federate save not in progress.
i) Federate restore not in progress.

10.29.4 Postconditions

a) The joined federate has the requested range bounds.

10.29.5 Exceptions

a) The region does not contain the specified dimension.
b) Invalid region handle.
c) Invalid region.
d) Invalid dimension handle.
e) Federate save in progress.
f) Federate restore in progress.
g) The federate is not a federation execution member.
h) Not connected.
i) RTI internal error.

10.29.6 Reference state charts

a) None.

10.30 *Set Range Bounds* service

The *Set Range Bounds* service shall set the current lower and upper bounds of the range for the specified dimension in the specified region.

10.30.1 Supplied arguments

a) Region handle.
b) Dimension handle.
c) Range lower bound.
d) Range upper bound.

10.30.2 Returned arguments

a) None.

10.30.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The region exists.
e) The region was created by the joined federate using the *Create Region* service.
f) The region contains the specified dimension.
g) The range lower bound is greater than or equal to zero.
h) The range upper bound is less than or equal to the specified dimension's upper bound.
i) The range lower bound is less than the range upper bound.

j) Federate save not in progress.
k) Federate restore not in progress.

10.30.4 Postconditions

a) The specified range is set to the specified values.

10.30.5 Exceptions

a) Invalid range bound.
b) Invalid region handle.
c) The region does not contain the specified dimension.
d) The region was not created by the joined federate using the *Create Region* service.
e) Invalid region.
f) Invalid dimension handle.
g) Federate save in progress.
h) Federate restore in progress.
i) The federate is not a federation execution member.
j) Not connected.
k) RTI internal error.

10.30.6 Reference state charts

a) None.

10.31 *Normalize Federate Handle* service

The *Normalize Federate Handle* service shall return a normalized value of the provided federate handle that can be used in constructing a region with the dimension HLAfederate.

10.31.1 Supplied arguments

a) Federate handle.

10.31.2 Returned arguments

a) Normalized value.

10.31.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The invoking federate is joined to that federation execution.

10.31.4 Postconditions

a) The normalized value corresponds to the provided federate handle.

10.31.5 Exceptions

a) Invalid federate handle.
b) The federate is not a federation execution member.
c) Not connected.

d) RTI internal error.

10.31.6 Reference state charts

a) None.

10.32 *Normalize Service Group* service

The *Normalize Service Group* service shall return a normalized value of the provided service group indicator that can be used in constructing a region with the dimension HLAserviceGroup.

10.32.1 Supplied arguments

a) Service group indicator.

10.32.2 Returned arguments

a) Normalized value.

10.32.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The invoking federate is joined to that federation execution.

10.32.4 Postconditions

a) The normalized value corresponds to the provided service group indicator.

10.32.5 Exceptions

a) Invalid service group.
b) The federate is not a federation execution member.
c) Not connected.
d) RTI internal error.

10.32.6 Reference state charts

a) None.

10.33 *Enable Object Class Relevance Advisory Switch* service

The *Enable Object Class Relevance Advisory Switch* service shall set the Object Class Relevance Advisory Switch on.

10.33.1 Supplied arguments

a) None.

10.33.2 Returned arguments

a) None.

10.33.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The Object Class Relevance Advisory Switch is off.
e) Federate save not in progress.
f) Federate restore not in progress.

10.33.4 Postconditions

a) The class relevance advisory switch is turned on.

10.33.5 Exceptions

a) The Object Class Relevance Advisory Switch is on.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

10.33.6 Reference state charts

a) None.

10.34 *Disable Object Class Relevance Advisory Switch* service

The *Disable Object Class Relevance Advisory Switch* service shall set the Object Class Relevance Advisory Switch off.

10.34.1 Supplied arguments

a) None.

10.34.2 Returned arguments

a) None.

10.34.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The Object Class Relevance Advisory Switch is on.
e) Federate save not in progress.
f) Federate restore not in progress.

10.34.4 Postconditions

a) The class relevance advisory switch is turned off.

10.34.5 Exceptions

a) The Object Class Relevance Advisory Switch is off.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

10.34.6 Reference state charts

a) None.

10.35 *Enable Attribute Relevance Advisory Switch* service

The *Enable Attribute Relevance Advisory Switch* service shall set the Attribute Relevance Advisory Switch on.

10.35.1 Supplied arguments

a) None.

10.35.2 Returned arguments

a) None.

10.35.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The Attribute Relevance Advisory Switch is off.
e) Federate save not in progress.
f) Federate restore not in progress.

10.35.4 Postconditions

a) The Attribute Relevance Advisory Switch is turned on.

10.35.5 Exceptions

a) The Attribute Relevance Advisory Switch is on.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

10.35.6 Reference state charts

a) Figure 15.

10.36 *Disable Attribute Relevance Advisory Switch* service

The *Disable Attribute Relevance Advisory Switch* service shall set the Attribute Relevance Advisory Switch off.

10.36.1 Supplied arguments

a) None.

10.36.2 Returned arguments

a) None.

10.36.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The Attribute Relevance Advisory Switch is on.
e) Federate save not in progress.
f) Federate restore not in progress.

10.36.4 Postconditions

a) The Attribute Relevance Advisory Switch is turned off.

10.36.5 Exceptions

a) The Attribute Relevance Advisory Switch is off.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

10.36.6 Reference state charts

a) Figure 15.

10.37 *Enable Attribute Scope Advisory Switch* service

The *Enable Attribute Scope Advisory Switch* service shall set the Attribute Scope Advisory Switch on.

10.37.1 Supplied arguments

a) None.

10.37.2 Returned arguments

a) None.

10.37.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The Attribute Scope Advisory Switch is off.
e) Federate save not in progress.
f) Federate restore not in progress.

10.37.4 Postconditions

a) The Attribute Scope Advisory Switch is turned on.

10.37.5 Exceptions

a) The Attribute Scope Advisory Switch is on.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

10.37.6 Reference state charts

a) Figure 15.

10.38 *Disable Attribute Scope Advisory Switch* service

The *Disable Attribute Scope Advisory Switch* service shall set the Attribute Scope Advisory Switch off.

10.38.1 Supplied arguments

a) None.

10.38.2 Returned arguments

a) None.

10.38.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The Attribute Scope Advisory Switch is on.
e) Federate save not in progress.
f) Federate restore not in progress.

10.38.4 Postconditions

a) The Attribute Scope Advisory Switch is turned off.

10.38.5 Exceptions

a) The Attribute Scope Advisory Switch is off.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

10.38.6 Reference state charts

a) Figure 15.

10.39 *Enable Interaction Relevance Advisory Switch* service

The *Enable Interaction Relevance Advisory Switch* service shall set the Interaction Relevance Advisory Switch on.

10.39.1 Supplied arguments

a) None.

10.39.2 Returned arguments

a) None.

10.39.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The Interaction Relevance Advisory Switch is off.
e) Federate save not in progress.
f) Federate restore not in progress.

10.39.4 Postconditions

a) The Interaction Relevance Advisory Switch is turned on.

10.39.5 Exceptions

a) The Interaction Relevance Advisory Switch is on.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

10.39.6 Reference state charts

a) Figure 13.

10.40 *Disable Interaction Relevance Advisory Switch* service

The *Disable Interaction Relevance Advisory Switch* service shall set the Interaction Relevance Advisory Switch off.

10.40.1 Supplied arguments

a) None.

10.40.2 Returned arguments

a) None.

10.40.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) The federate is joined to that federation execution.
d) The Interaction Relevance Advisory Switch is on.
e) Federate save not in progress.
f) Federate restore not in progress.

10.40.4 Postconditions

a) The Interaction Relevance Advisory Switch is turned off.

10.40.5 Exceptions

a) The Interaction Relevance Advisory Switch is off.
b) Federate save in progress.
c) Federate restore in progress.
d) The federate is not a federation execution member.
e) Not connected.
f) RTI internal error.

10.40.6 Reference state charts

a) Figure 13.

10.41 *Evoke Callback* service

The *Evoke Callback* service instructs the RTI that the invoking federate is prepared to receive a single federate callback. If there are no pending callbacks to be delivered to the federate, this service invocation shall wait for the duration indicated by the supplied argument. The supplied argument shall have a precision of at least 1 ms. This service shall result in either the timeout expiring or a single callback to be invoked on the federate. The pending callback indicator argument shall be a boolean value: TRUE shall indicate that pending callbacks exist, and FALSE shall indicate that no pending callbacks exist. This service has no effect in the IMMEDIATE callback model.

10.41.1 Supplied arguments

a) Minimum amount of wall-clock time the service will wait for a callback.

10.41.2 Returned arguments

a) Pending callback indicator.

10.41.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.

10.41.4 Postconditions

a) The RTI is provided the opportunity to invoke callbacks on the federate.

10.41.5 Exceptions

a) The federate is not a federation execution member.
b) Not connected.
c) RTI internal error.

10.41.6 Reference state charts

a) None.

10.42 *Evoke Multiple Callbacks* service

The *Evoke Multiple Callbacks* service instructs the RTI that the invoking federate is prepared to receive multiple federate callbacks. The service shall continue to process available callbacks until the minimum specified wall-clock time. At that wall-clock time, if there are no additional callbacks to be delivered to the federate, the service shall complete. If, after the minimum specified wall-clock time, there continue to be callbacks, the RTI shall continue to deliver those callbacks until the maximum specified wall-clock time is exceeded. Both supplied wall-clock time arguments shall have a precision of at least 1 ms. The pending callback indicator argument shall be a boolean value: TRUE shall indicate that pending callbacks exist, and FALSE shall indicate that no pending callbacks exist. This service has no effect in the IMMEDIATE callback model.

10.42.1 Supplied arguments

a) Minimum amount of wall-clock time.
b) Maximum amount of wall-clock time.

10.42.2 Returned arguments

a) Pending callback indicator.

10.42.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.

10.42.4 Postconditions

a) The RTI is provided the opportunity to invoke callbacks on the federate.

10.42.5 Exceptions

a) The federate is not a federation execution member.
b) Not connected.
c) RTI internal error.

10.42.6 Reference state charts

a) None.

10.43 *Enable Callbacks* service

In the EVOKED callback model, the *Enable Callbacks* service instructs the RTI to deliver any available callbacks processed through the *Evoke Callback* and *Evoke Multiple Callbacks* services. In the IMMEDIATE callback model, the *Enable Callbacks* service instructs the RTI to deliver any available callbacks processed immediately when they are available. The default state for a federate is that callbacks are enabled.

10.43.1 Supplied arguments

a) None.

10.43.2 Returned arguments

a) None.

10.43.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) Federate save not in progress.
d) Federate restore not in progress.

10.43.4 Postconditions

a) The RTI shall process any pending callbacks on the federate when either the *Evoke Callback* or *Evoke Multiple Callbacks* service is invoked.

10.43.5 Exceptions

a) Federate save in progress.
b) Federate restore in progress.
c) The federate is not a federation execution member.
d) Not connected.
e) RTI internal error.

10.43.6 Reference state charts

a) None.

10.44 *Disable Callbacks* service

The *Disable Callbacks* service instructs the RTI to defer the delivery of any available callbacks. This service can be used to implement a guard mechanism to control when callbacks are delivered to the federate.

10.44.1 Supplied arguments

a) None.

10.44.2 Returned arguments

a) None.

10.44.3 Preconditions

a) The federate is connected to the RTI.
b) The federation execution exists.
c) Federate save not in progress.
d) Federate restore not in progress.

10.44.4 Postconditions

a) The RTI shall not process any pending callbacks on the federate for any existing or subsequent *Evoke Callback* or *Evoke Multiple Callbacks* invocations until the *Enable Callback* service is invoked.

10.44.5 Exceptions

a) Federate save in progress.
b) Federate restore in progress.
c) The federate is not a federation execution member.
d) Not connected.
e) RTI internal error.

10.44.6 Reference state charts

a) None.

11. Management object model (MOM)

11.1 Overview

The MOM provides facilities for access to RTI operating information during federation execution. These MOM facilities can be used by joined federates to gain insight into the operations of the federation execution and to control the functioning of the RTI, the federation execution, and individual joined federates. The ability to monitor and control elements of a federation is required for proper functioning of a federation execution.

Access and interchange of RTI information elements included in the MOM shall be accomplished by utilizing predefined HLA constructs, objects, and interactions in the same way that participating federates exchange information with other federates. The MOM shall use the object model template (OMT) format (found in IEEE Std 1516.2-2010) and syntax for the definition of these information and control elements. The RTI publishes object classes, registers object instances, and updates values of attributes of those object instances; subscribes to and receives some interactions; and publishes and sends other interactions. A joined federate charged with controlling a federation execution can subscribe to some or all of the object classes, reflect the updates, publish and send some interactions, and subscribe to and receive other interactions. Thus, while a FDD for a federation execution shall include all elements of the MOM, a federation may choose to use all, part, or none of the MOM standard classes and associated information elements. The MOM shall satisfy these requirements by utilizing predefined HLA constructs: object instances and interactions. The RTI shall publish object classes and shall register object instances and update values of attributes of those object instances; shall subscribe to and receive some interaction classes; and shall publish and send other interaction classes. A joined federate charged with controlling a federation execution can subscribe to the object classes, reflect the updates, publish and send some interaction classes, and subscribe to and receive other interaction classes.

All MOM object classes, interaction classes, attributes, and parameters shall be predefined in the FDD. These definitions shall not be revised. However, MOM definitions may be extended. They may be augmented with additional subclasses, class attributes, or parameters. These new elements shall not be acted upon directly by the RTI; they may be acted on by federates in the federation.

11.2 MOM object classes

The high-level MOM object class structure is depicted in Figure 19. The MOM object classes shall be defined as follows:

a) Object class HLAmanager.HLAfederate contains attributes that describe the state of a federate. The RTI shall publish the class and register one object instance of this class for each federate in the federation. The RTI shall update the information periodically, based on timing data provided in HLAmanager.HLAfederate.HLAadjust interactions. Information shall be contained in an object instance that includes identifying information about the federate, measures of the federate's state, and the status of the queued messages maintained by the RTI for the federate.
b) Object class HLAmanager.HLAfederation contains attributes that describe the state of the federation execution. The RTI shall publish the class and register one object instance of this class for the federation.

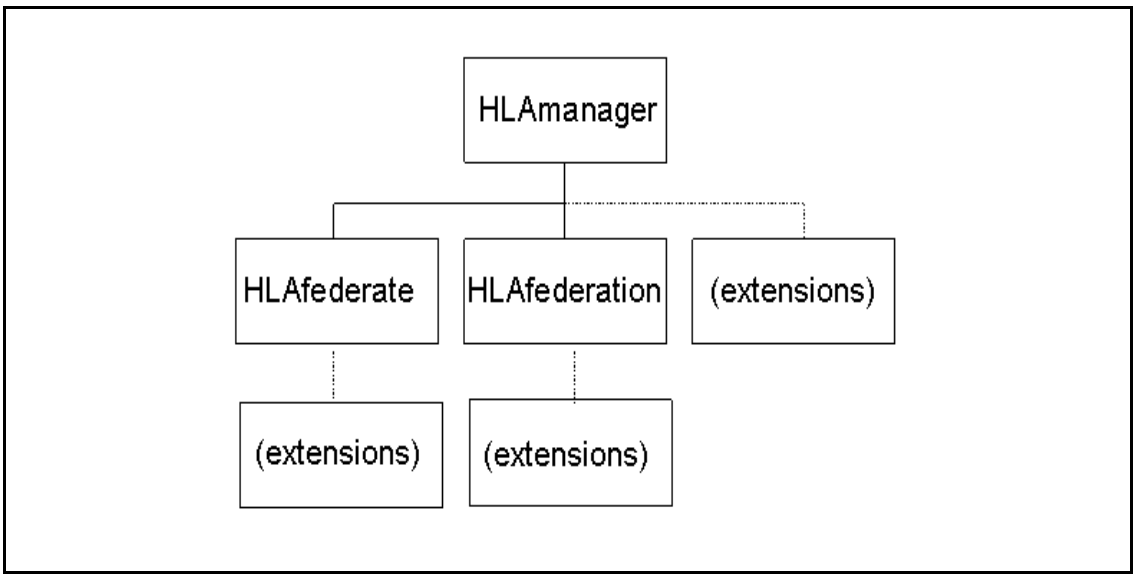

Figure 19—MOM object class structure

11.2.1 Extension of MOM object classes

The MOM object classes may be extended by adding subclasses or class attributes. The RTI shall publish HLAmanager.HLAfederate and HLAmanager.HLAfederation classes with predefined MOM class attributes, register an instance, and update the values of the predefined instance attributes.

The RTI shall not subscribe to any object class. Valid methods for extending the MOM object classes shall be as follows:

a) Subclasses may be added to any MOM object class. Here, the joined federate may publish the object class and its attributes, register an instance of the new class, and update values of instance attributes of the object instance according to the dictates of the federation execution. Note that the instance of the subclass shall be separate from the MOM object instance that is registered by the RTI. Therefore, instance attributes that are inherited by the extension subclass from the MOM predefined class shall not be updated by the RTI.
b) Attributes may be added to any MOM object class. Here, the joined federate may publish the object class with the new class attributes, may subscribe to the object class and attributes in it, may discover and reflect updates to learn the object instance in question, and may update the values of the new instance attributes (after acquiring ownership) using the discovered object instance designator. Note that the instance that the joined federate shall update with the new instance attributes shall be the same as the MOM object instance that is registered by the RTI. The RTI shall never acquire ownership of any new class attributes.

The RTI shall not divest ownership of MOM instance attributes that are defined in this clause. The RTI shall respond to the invocation, by any federate, of the *Request Attribute Value Update* service for any MOM object class or instance attribute that is defined in this clause by supplying values via the normal instance attribute update mechanism.

11.3 MOM interaction classes

The high-level MOM interaction class structure is depicted in Figure 20. The MOM interaction classes shall be defined as follows:

a) Interaction classes that are subclasses of HLAmanager.HLAfederate.HLAadjust and HLAmanager.Hlafederation.HLAadjust shall be acted on by the RTI. They permit a managing federate to adjust how the RTI performs when responding to another federate and how it shall respond and report to the managing federate.
b) Interaction classes that are subclasses of HLAmanager.HLAfederate.HLArequest shall be acted on by the RTI. They shall cause the RTI to send subclasses of HLAmanager.HLAfederate.HLAreport interaction class.
c) Interaction classes that are subclasses of HLAmanager.HLAfederate.HLAreport shall be sent by the RTI. They shall respond to interaction classes that are subclasses of HLAmanager.HLAfederate.HLArequest class interactions. They shall describe some aspect of the federate, such as its object class subscription tree.
d) Interaction classes that are subclasses of HLAmanager.HLAfederate.HLAservice shall be acted on by the RTI. They shall invoke HLA services on behalf of another federate. These interactions shall cause the RTI to react as if the service was invoked by that other federate (for example, a managing federate could change the time-regulating state of another federate).

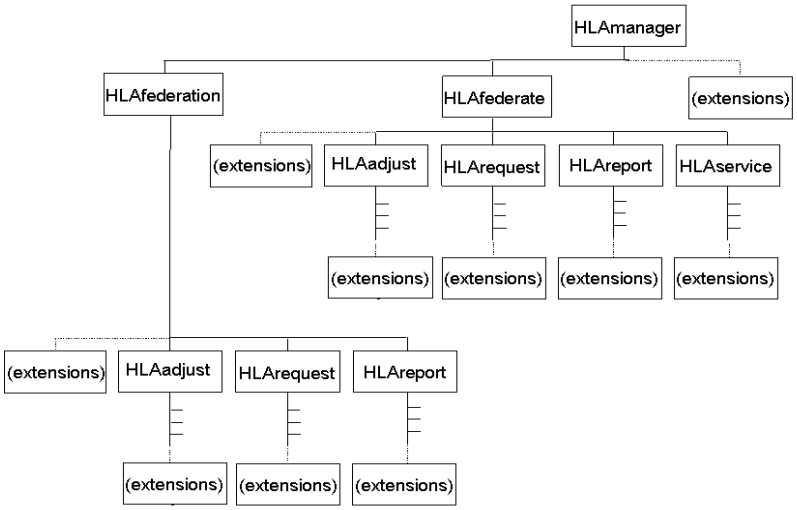

Figure 20—MOM interaction class structure

11.3.1 Extension of MOM interaction classes

The MOM interaction classes may be extended by adding subclasses or parameters. There are three categories of extension of MOM interaction classes as follows:

a) Classes of interaction that the RTI shall send (subclasses of HLAmanager.HLAfederate.HLAreport). The RTI shall publish at the MOM leaf-class level (e.g., HLAmanager.HLAfederate.HLA-report.HLAreportException). It shall send interactions containing all predefined parameters for that interaction class. Valid methods for extending this type of MOM interaction class are as follows:
    1) Subclasses may be added to these MOM interaction classes. The RTI shall not send interactions of these subclasses. If joined federates subscribe to the subclass, they shall receive the full interaction. If they subscribe to the class of which the extension is a subclass, the interaction shall be promoted to the subscribed class, and any new parameters shall be lost.
    2) Parameters may be added to any MOM interaction class. Interactions of these classes that are sent by the RTI shall not contain the new parameters.

b) Classes of interaction that the RTI shall receive (subclasses of HLAmanager.HLAfederate.HLA-adjust, HLAmanager.HLAfederate.HLArequest, HLAmanager.HLAfederate.HLAservice, and HLAmanager.HLAfederation.HLAadjust). The RTI shall subscribe at the MOM leaf-class level (e.g., HLAmanager.HLAfederate.HLAadjust.HLAsetTiming). It shall receive these interactions and process all predefined parameters for that interaction class (with exceptions as listed in the MOM Parameters Definition Table). Valid methods for extending this type of MOM interaction class are as follows:
   1) Subclasses may be added to any MOM interaction class. If a joined federate sends an interaction of this class, the RTI shall receive a promoted version that shall contain only the parameters of the predefined interaction class.
   2) Parameters may be added to any MOM interaction class. If a joined federate sends an interaction with extra parameters, the RTI shall receive the new parameters, but shall ignore them and process only the predefined parameters.
c) Classes of interaction that shall be neither sent nor received by the RTI. These classes of interaction shall be ignored by the RTI and may be formed in any way that is consistent with IEEE 1516.2 federation object model (FOM) development.

11.4 MOM-related characteristics of the RTI

11.4.1 Normal RTI MOM administration

The RTI deals with MOM object classes and interaction classes in the following ways:

a) The RTI shall publish all leaf object classes in Table 4 that are designated with a "P."
b) For each leaf object class in Table 4, the RTI shall publish all attributes in Table 6 for that object class that are designated with a "P," and no more.
c) The RTI shall publish all leaf interaction classes in Table 5 that are designated with a "P."
d) In response to an interaction of class HLAmanager.HLAfederate.HLArequest where the requested information does not exist at the designated federate, the RTI shall send the corresponding interaction of class HLAmanager.HLAfederate.HLAreport with the appropriate values that indicate a NULL response. The conditions for a NULL response are given in Table 15. The appropriate value and inclusion of a parameter in a NULL response is given in Table 17.
   1) The RTI may be required to provide an attribute or parameter whose value is undefined. For an attribute or parameter having an array value, an undefined value shall be encoded as an empty (zero-length) array.
e) When sending an interaction of one of the leaf classes in Table 5 that is not a NULL response, the RTI shall always supply all parameters listed in Table 7 for that interaction class, and no more.
f) The RTI shall be actively subscribed to all leaf interaction classes in Table 5 that are designated with an "S."
g) The RTI shall be neither time-regulating nor time-constrained and shall never provide timestamps or user-supplied tags with instance attribute updates, sent interactions, or deletion of object instances.
h) The RTI shall update the values of all attributes of instances of HLAmanager.HLAfederate found in Table 6 using an update region set that contains a single region. The region used with each instance attribute shall contain only the HLAfederate dimension as a specified dimension. The range for that dimension shall be a point range around the normalized value (using the *Normalize Federate Handle* service) of the HLAfederateHandle attribute of that object instance.

i) The RTI shall send all interactions of class HLAmanager.HLAfederate.HLAreport.HLAreport-ServiceInvocation using an update region set that contains a single region. The region used with each such interaction shall contain only the HLAfederate and HLAserviceGroup dimensions as specified dimensions. The range used for the HLAfederate dimension shall be a point range around the normalized value (using the *Normalize Federate Handle* service) of the HLAfederate parameter of that interaction. The range used for the HLAserviceGroup dimension shall be a point range around the normalized value (using the *Normalize Service Group* service) of the service group in which the service indicated in the HLAservice parameter is found.

j) The RTI shall send all other interactions that are of subclasses of HLAmanager.HLA-federate.HLAreport in Table 5 using an update region set that contains a single region. The region used with each such interaction shall contain only the HLAfederate dimension as a specified dimension. The range for that dimension shall be a point range around the normalized value (using the *Normalize Federate Handle* service) of the HLAfederate parameter of that interaction.

k) Other than registering and deleting object instances, updating instance attributes, and sending and receiving interactions, the RTI shall not engage in any other activity with regard to MOM. In other words, among other things, an RTI shall never
   1) Engage in ownership transfer of MOM instance attributes.
   2) Modify the transportation or order type of a MOM instance attribute or interaction class.
   3) Use DDM services with MOM instance attributes or interaction classes, except as noted above.

l) 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:
   1) A federate shall not be required to supply parameters of any HLAmanager.HLA-federate.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.)
   2) A federate shall supply at least one of the parameters of the HLAmanager.HLAfeder-ate.HLAadjust.HLAsetSwitches interaction.

11.4.2 Sending MOM interactions by federates during federate save/restore

Although sending interactions is normally forbidden while federate saves and restores are in progress, the following deviations to this rule shall be allowed (assuming the correct publications are in place):

a) A joined federate shall be allowed to send either a MOM HLAfederateSaveBegun or a MOM HLAfederateSaveComplete interaction when a federate save is in progress.

b) A joined federate shall be allowed to send a MOM HLAfederateRestoreComplete interaction when a federate restore is in progress.

11.5 Service reporting

The HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation interaction shall be sent by the RTI whenever an HLA service is invoked, either by a particular joined federate or by the RTI at a particular joined federate, and service reporting switch is enabled for that particular joined federate. The service reporting switch value for a specific joined federate shall be determined in the following manner:

a) The initial value for the service reporting switch for that joined federate shall be the value of the service reporting switch in the federation execution's FDD (enabled or disabled).
b) If one or more HLAmanager.HLAfederate.HLAadjust.HLAsetSwitches interactions have set the service reporting switch for that joined federate, the value shall be that set by the last interaction (enabled or disabled).

If a joined federate is subscribed to the HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation interaction, all attempts to set the service reporting switch for that joined federate to enabled by sending an HLAmanager.HLAfederate.HLAadjust.HLAsetSwitches interaction shall fail and be reported via the normal MOM interaction failure means. If a joined federate has the Service Reporting Switch Enabled, that joined federate shall not be able to subscribe to the HLAmanager.HLAfederate.HLAreport.HLAreportService-Invocation interaction class.

11.6 MOM OMT tables

The elements that constitute MOM are depicted in OMT format (found in IEEE Std 1516.2-2010) in Table 4 through Table 17. The MOM table information in this clause shall appear in all compliant FOMs (see IEEE Std 1516-2010 and IEEE Std 1516.2-2010) and shall not be altered. Note that MOM does not use many of the OMT tables (e.g., time representation table, user-supplied tag table, synchronization table, transportation type table, switches table, and variant record table). They have been omitted from the depiction.

If there are any differences between the MOM tables defined in this clause (i.e., Clause 11) and the tables defined in the MOM and Initialization Module (MIM) in Annex G, this clause shall take precedence.

Table 4—MOM object class structure table

HLAobjectRoot (N)	HLAmanager (N)	HLAfederate (P)
		HLAfederation (P)

Table 5—MOM interaction class structure table

HLAinteractionRoot (N)	HLAmanager (N)	HLAfederate (N)	HLAadjust (N)	HLAsetTiming (S)
				HLAmodifyAttributeState (S)
				HLAsetSwitches (S)
			HLArequest (N)	HLArequestPublications (S)
				HLArequestSubscriptions (S)
				HLArequestObjectInstancesThatCanBeDeleted (S)
				HLArequestObjectInstancesUpdated (S)
				HLArequestObjectInstancesReflected (S)
				HLArequestUpdatesSent (S)
				HLArequestInteractionsSent (S)
				HLArequestReflectionsReceived (S)
				HLArequestInteractionsReceived (S)
				HLArequestObjectInstanceInformation (S)
				HLArequestFOMmoduleData (S)
			HLAreport (N)	HLAreportObjectClassPublication (P)
				HLAreportObjectClassSubscription (P)
				HLAreportInteractionPublication (P)
				HLAreportInteractionSubscription (P)
				HLAreportObjectInstancesThatCanBeDeleted (P)
				HLAreportObjectInstancesUpdated (P)
				HLAreportObjectInstancesReflected (P)
				HLAreportUpdatesSent (P)
				HLAreportReflectionsReceived (P)
				HLAreportInteractionsSent (P)
				HLAreportInteractionsReceived (P)
				HLAreportObjectInstanceInformation (P)
				HLAreportException (P)
				HLAreportServiceInvocation (P)
				HLAreportMOMexception (P)
				HLAreportFederateLost (P)
				HLAreportFOMmoduleData (P)
			HLAservice (N)	HLAresignFederationExecution (S)
				HLAsynchronizationPointAchieved (S)
				HLAfederateSaveBegun (S)
				HLAfederateSaveComplete (S)
				HLAfederateRestoreComplete (S)
				HLApublishObjectClassAttributes (S)
				HLAunpublishObjectClassAttributes (S)
				HLApublishInteractionClass (S)
				HLAunpublishInteractionClass (S)
				HLAsubscribeObjectClassAttributes (S)

Table 5—MOM interaction class structure table *(continued)*

				HLAunsubscribeObjectClassAttributes (S)
				HLAsubscribeInteractionClass (S)
				HLAunsubscribeInteractionClass (S)
				HLAdeleteObjectInstance (S)
				HLAlocalDeleteObjectInstance (S)
				HLArequestAttributeTransportationypeChange (S)
				HLArequestInteractionTransportationype-Change (S)
				HLAunconditionalAttributeOwnershipDives-titure (S)
				HLAenableTimeRegulation (S)
				HLAdisableTimeRegulation (S)
				HLAenableTimeConstrained (S)
				HLAdisableTimeConstrained (S)
				HLAtimeAdvanceRequest (S)
				HLAtimeAdvanceRequestAvailable (S)
				HLAnextMessageRequest (S)
				HLAnextMessageRequestAvailable (S)
				HLAflushQueueRequest (S)
				HLAenableAsynchronousDelivery (S)
				HLAdisableAsynchronousDelivery (S)
				HLAmodifyLookahead (S)
				HLAchangeAttributeOrderType (S)
				HLAchangeInteractionOrderType (S)
		HLAfederation (N)	HLAadjust (N)	HLAsetSwitches (S)
			HLArequest (N)	HLArequestSynchronizationPoints (S)
				HLArequestSynchronizationPointStatus (S)
				HLArequestFOMmoduleData (S)
				HLArequestMIMData (S)
			HLAreport (N)	HLAreportSynchronizationPoints (P)
				HLAreportSynchronizationPointStatus (P)
				HLAreportFOMmoduleData (P)
				HLAreportMIMData (P)

Table 6—MOM attribute table

Object	Attribute	Datatype	Update type	Update condition	D/A	P/S	Available dimensions	Transportation	Order
HLAobjectRoot. HLAmanager.HLAfederate	HLAfederateHandle	HLAhandle	Static	NA	N	P	HLAfederate	HLAreliable	Receive
	HLAfederateName	HLAunicodeString	Static	NA	N	P	HLAfederate	HLAreliable	Receive
	HLAfederateType	HLAunicodeString	Static	NA	N	P	HLAfederate	HLAreliable	Receive
	HLAfederateHost	HLAunicodeString	Static	NA	N	P	HLAfederate	HLAreliable	Receive
	HLARTIversion	HLAunicodeString	Static	NA	N	P	HLAfederate	HLAreliable	Receive
	HLAFOMmoduleDesignatorList	HLAmoduleDesignatorlist	Static	N/A	N	P	HLAfederate	HLAreliable	Receive
	HLAtimeConstrained	HLAboolean	Conditional	Whenever services *Time Constrained Enabled* † or *Disable Time Constrained* are successfully invoked (including via the HLAdisableTimeConstrained interaction).	N	P	HLAfederate	HLAreliable	Receive
	HLAtimeRegulating	HLAboolean	Conditional	Whenever services *Time Regulation Enabled* † or *Disable Time Regulation* are successfully invoked (including via the HLAdisableTimeRegulation interaction).	N	P	HLAfederate	HLAreliable	Receive
	HLAasynchronousDelivery	HLAboolean	Conditional	Whenever services *Enable Asynchronous Delivery* or *Disable Asynchronous Delivery* are successfully invoked (including via the HLAenableAsynchronousDelivery or HLAdisableAsynchronousDelivery interactions).	N	P	HLAfederate	HLAreliable	Receive

Table 6—MOM attribute table *(continued)*

Object	Attribute	Datatype	Update type	Update condition	D/A	P/S	Available dimensions	Transportation	Order
	HLAfederateState	HLAfederateState	Conditional	Whenever the services *Initiate Federate Save †*, *Federation Saved †*, *Federation Restore Begun †*, *Confirm Federation Restoration Request (success) †*, or *Join Federation Execution* are successfully invoked. Also, after the *Federation Restored †* service has been invoked at all federates in the federation execution. If a joined federate is in the Federate Save in Progress state, no corresponding reflects shall be invoked at that joined federate.	N	P	HLAfederate	HLAreliable	Receive
	HLAtimeManagerState	HLAtimeState	Conditional	Whenever services *Time Advance Request*, *Time Advance Request Available*, *Next Message Request, Next Message Request Available*, *Flush Queue Request*, or *Time Advance Grant †* are successfully invoked (including via the HLAtimeAdvanceRequest, HLAtimeAdvanceRequest-Available, HLAnext-MessageRequest, HLAnextMessageRequest-Available, or HLAflush-QueueRequest interactions).	N	P	HLAfederate	HLAreliable	Receive
	HLAlogicalTime	HLAlogicalTime	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive
	HLAlookahead	HLAtimeInterval	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive
	HLAGALT	HLAlogicalTime	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive

Table 6—MOM attribute table *(continued)*

Object	Attribute	Datatype	Update type	Update condition	D/A	P/S	Available dimensions	Transportation	Order
	HLALITS	HLAlogicalTime	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive
	HLAROlength	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive
	HLATSOlength	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive
	HLAreflectionsReceived	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive
	HLAupdatesSent	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive
	HLAinteractionsReceived	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive
	HLAinteractionsSent	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive
	HLAobjectInstancesThat-CanBeDeleted	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive
	HLAobjectInstancesUpdated	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive
	HLAobjectInstances-Reflected	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive
	HLAobjectInstancesDeleted	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive
	HLAobjectInstances-Removed	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive
	HLAobjectInstances-Registered	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive
	HLAobjectInstances-Discovered	HLAcount	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive
	HLAtimeGrantedTime	HLAmsec	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive
	HLAtimeAdvancingTime	HLAmsec	Periodic	HLAsetTiming. HLAreportPeriod	N	P	HLAfederate	HLAreliable	Receive

Table 6—MOM attribute table *(continued)*

Object	Attribute	Datatype	Update type	Update condition	D/A	P/S	Available dimensions	Transportation	Order
	HLAconveyRegionDesignatorSets	HLAswitch	Conditional	Whenever the HLAmanager.HLAfederate.HLAfederate.HLA-adjust.HLAsetSwitches interaction is sent to successfully change the value of the HLAconvey-RegionDesignatorSets parameter.	N	P	HLAfederate	HLAreliable	Receive
	HLAconveyProducing-Federate	HLAswitch	Conditional	Whenever the HLAmanager.HLAfederate.HLAadjust.HLAsetSwitches interaction is sent to successfully change the value of the HLAconveyProducing-Federate parameter.	N	P	HLAfederate	HLAreliable	Receive
HLAmanager.HLAfederation	HLAfederationName	HLAunicodeString	Static	NA	N	P	NA	HLAreliable	Receive
	HLAfederatesInFederation	HLAhandleList	Conditional	Federate joins or resigns	N	P	NA	HLAreliable	Receive
	HLARTIversion	HLAunicodeString	Static	NA	N	P	NA	HLAreliable	Receive
	HLAMIMDesignator	HLAunicodeString	Static	NA	N	P	NA	HLAreliable	Receive
	HLAFOMmoduleDesignatorList	HLAmoduleDesignatorlist	Conditional	On change	N	P	NA	HLAreliable	Receive
	HLAcurrentFDD	HLAunicodeString	Conditional	On change	N	P	NA	HLAreliable	Receive
	HLAtimeImplementation-Name	HLAunicodeString	Static	NA	N	P	NA	HLAreliable	Receive
	HLAlastSaveName	HLAunicodeString	Conditional	Whenever *Federation Saved* service is successfully invoked with a save-success indicator of successful	N	P	NA	HLAreliable	Receive
	HLAlastSaveTime	HLAlogicalTime	Conditional	Whenever *Federation Saved* service is successfully invoked with a save-success indicator of successful	N	P	NA	HLAreliable	Receive
	HLAnextSaveName	HLAunicodeString	Conditional	Whenever *Request Federation Save* service is successfully invoked	N	P	NA	HLAreliable	Receive

Table 6—MOM attribute table *(continued)*

Object	Attribute	Datatype	Update type	Update condition	D/A	P/S	Available dimensions	Transportation	Order
	HLAnextSaveTime	HLAlogicalTime	Conditional	Whenever *Request Federation Save* service is successfully invoked	N	P	NA	HLAreliable	Receive
	HLAautoProvide	HLAswitch	Conditional	Whenever the HLAmanager.HLAfederate.HLAfederation.HLAadjust.HLAsetSwitches interaction is sent to successfully change the value of the HLAautoProvide parameter.	N	P	NA	HLAreliable	Receive

Table 7—MOM parameter table

Interaction		Parameter	Datatype	Available dimensions	Transportation	Order
HLAinteractionRoot	HLAmanager	NA	NA	NA	HLAreliable	Receive
HLAmanager	HLAfederate	HLAfederate	HLAhandle	NA	HLAreliable	Receive
HLAmanager.HLAfederate	HLAadjust	NA	NA	NA	HLAreliable	Receive
HLAmanager.HLAfederate.HLA-adjust	HLAsetTiming	HLAreportPeriod	HLAseconds	NA	HLAreliable	Receive
	HLAmodifyAttributeState	HLAobjectInstance	HLAhandle	NA	HLAreliable	Receive
		HLAattribute	HLAhandle			
		HLAattributeState	HLAownership			
	HLAsetSwitches	HLAconveyRegion-DesignatorSets	HLAswitch	NA	HLAreliable	Receive
		HLAconveyProduc-ingFederate	HLAswitch			Receive
		HLAserviceReporting	HLAswitch			Receive
		HLAexceptionRe-porting	HLAswitch			Receive

Table 7—MOM parameter table *(continued)*

Interaction		Parameter	Datatype	Available dimensions	Transportation	Order
HLAmanager.HLAfederate	HLArequest	NA	NA	NA	HLAreliable	Receive
HLAmanager.HLAfederate.HLA-request	HLArequestPublications	NA	NA	NA	HLAreliable	Receive
	HLArequestSubscriptions	NA	NA	NA	HLAreliable	Receive
	HLArequestObjectInstancesThatCan-BeDeleted	NA	NA	NA	HLAreliable	Receive
	HLArequestObjectInstancesUpdated	NA	NA	NA	HLAreliable	Receive
	HLArequestObjectInstancesReflected	NA	NA	NA	HLAreliable	Receive
	HLArequestUpdatesSent	NA	NA	NA	HLAreliable	Receive
	HLArequestInteractionsSent	NA	NA	NA	HLAreliable	Receive
	HLArequestReflectionsReceived	NA	NA	NA	HLAreliable	Receive
	HLArequestInteractionsReceived	NA	NA	NA	HLAreliable	Receive
	HLArequestObjectInstanceInformation	HLAobjectInstance	HLAhandle	NA	HLAreliable	Receive
	HLArequestFOMmoduleData	HLAFOMmodule-Indicator	HLAindex	NA	HLAreliable	Receive
HLAmanager.HLAfederate	HLAreport	NA	NA	NA	HLAreliable	Receive
HLAmanager.HLAfederate.HLA-report	HLAreportObjectClassPublication	HLAnumberOfClasses	HLAcount	HLAfederate	HLAreliable	Receive
		HLAobjectClass	HLAhandle			
		HLAattributeList	HLAhandleList			
	HLAreportObjectClassSubscription	HLAnumberOfClasses	HLAcount	HLAfederate	HLAreliable	Receive
		HLAobjectClass	HLAhandle			
		HLAactive	HLAboolean			
		HLAmaxUpdateRate	HLAupdateRateName			
		HLAattributeList	HLAhandleList			
	HLAreportInteractionPublication	HLAinteractionClass-List	HLAhandleList	HLAfederate	HLAreliable	Receive
	HLAreportInteractionSubscription	HLAinteractionClass-List	HLAinteractionSubList	HLAfederate	HLAreliable	Receive
	HLAreportObjectInstancesThatCanBe-Deleted	HLAobjectInstance-Counts	HLAobjectClassBasedCounts	HLAfederate	HLAreliable	Receive

Table 7—MOM parameter table *(continued)*

Interaction		Parameter	Datatype	Available dimensions	Transportation	Order
	HLAreportObjectInstancesUpdated	HLAobjectInstance-Counts	HLAobjectClassBasedCounts	HLAfederate	HLAreliable	Receive
	HLAreportObjectInstancesReflected	HLAobjectInstance-Counts	HLAobjectClassBasedCounts	HLAfederate	HLAreliable	Receive
	HLAreportUpdatesSent	HLAtransportation	HLAtransportationName	HLAfederate	HLAreliable	Receive
		HLAupdateCounts	HLAobjectClassBasedCounts			
	HLAreportReflectionsReceived	HLAtransportation	HLAtransportationName	HLAfederate	HLAreliable	Receive
		HLAreflectCounts	HLAobjectClassBasedCounts			
	HLAreportInteractionsSent	HLAtransportation	HLAtransportationName	HLAfederate	HLAreliable	Receive
		HLAinteractionCounts	HLAinteractionCounts			
	HLAreportInteractionsReceived	HLAtransportation	HLAtransportationName	HLAfederate	HLAreliable	Receive
		HLAinteractionCounts	HLAinteractionCounts			
HLAmanager.HLAfederate.HLA-report	HLAreportObjectInstanceInformation	HLAobjectInstance	HLAhandle	HLAfederate	HLAreliable	Receive
		HLAownedInstance-AttributeList	HLAhandleList			
		HLAregisteredClass	HLAhandle			
		HLAknownClass	HLAhandle			
	HLAreportException	HLAservice	HLAunicodeString	HLAfederate	HLAreliable	Receive
		HLAexception	HLAunicodeString			
	HLAreportServiceInvocation	HLAservice	HLAunicodeString	HLAfederate HLAserviceGroup	HLAreliable	Receive
		HLAsuccessIndicator	HLAboolean			
		HLAsuppliedArguments	HLAargumentList			
		HLAreturnedArguments	HLAargumentList			
		HLAexception	HLAunicodeString			
		HLAserialNumber	HLAcount			

Table 7—MOM parameter table *(continued)*

Interaction		Parameter	Datatype	Available dimensions	Transportation	Order
	HLAreportMOMexception	HLAservice	HLAunicodeString	HLAfederate	HLAreliable	Receive
		HLAexception	HLAunicodeString			
		HLAparameterError	HLAboolean			
	HLAreportFederateLost	HLAfederateName	HLAunicodeString	HLAfederate	HLAreliable	Receive
		HLAtimestamp	HLAlogicalTime			
		HLAfaultDescription	HLAunicodeString			
	HLAreportFOMmoduleData	HLAFOMmodule-Indicator	HLAindex	HLAfederate	HLAreliable	Receive
		HLAFOMmoduleData	HLAunicodeString			
HLAmanager.HLAfederate	HLAservice	NA	NA	NA	HLAreliable	Receive
HLAmanager.HLAfederate.HLA-service	HLAresignFederationExecution	HLAresignAction	HLAresignAction	NA	HLAreliable	Receive
	HLAsynchronizationPointAchieved	HLAlabel	HLAunicodeString	NA	HLAreliable	Receive
	HLAfederateSaveBegun	NA	NA	NA	HLAreliable	Receive
HLAmanager.HLAfederate.HLA-service	HLAfederateSaveComplete	HLAsuccessIndicator	HLAboolean	NA	HLAreliable	Receive
	HLAfederateRestoreComplete	HLAsuccessIndicator	HLAboolean	NA	HLAreliable	Receive
	HLApublishObjectClassAttributes	HLAobjectClass	HLAhandle	NA	HLAreliable	Receive
		HLAattributeList	HLAhandleList			
	HLAunpublishObjectClassAttributes	HLAobjectClass	HLAhandle	NA	HLAreliable	Receive
		HLAattributeList	HLAhandleList			
	HLApublishInteractionClass	HLAinteractionClass	HLAhandle	NA	HLAreliable	Receive
	HLAunpublishInteractionClass	HLAinteractionClass	HLAhandle	NA	HLAreliable	Receive
	HLAsubscribeObjectClassAttributes	HLAobjectClass	HLAhandle	NA	HLAreliable	Receive
		HLAattributeList	HLAhandleList			
		HLAactive	HLAboolean			
	HLAunsubscribeObjectClassAttributes	HLAobjectClass	HLAhandle	NA	HLAreliable	Receive
		HLAattributeList	HLAhandleList			

Table 7—MOM parameter table *(continued)*

Interaction		Parameter	Datatype	Available dimensions	Transportation	Order
	HLAsubscribeInteractionClass	HLAinteractionClass	HLAhandle	NA	HLAreliable	Receive
		HLAactive	HLAboolean			
	HLAunsubscribeInteractionClass	HLAinteractionClass	HLAhandle	NA	HLAreliable	Receive
	HLAdeleteObjectInstance	HLAobjectInstance	HLAhandle	NA	HLAreliable	Receive
		HLAtag	HLAopaqueData			
		HLAtimeStamp	HLAlogicalTime			
	HLAlocalDeleteObjectInstance	HLAobjectInstance	HLAhandle	NA	HLAreliable	Receive
	HLArequestAttributeTransportation-TypeChange	HLAobjectInstance	HLAhandle	NA	HLAreliable	Receive
		HLAattributeList	HLAhandleList			
		HLAtransportation	HLAtransportationName			
	HLArequestInteractionTransportation-TypeChange	HLAinteractionClass	HLAhandle	NA	HLAreliable	Receive
		HLAtransportation	HLAtransportationName			
HLAmanager.HLAfederate.HLA-service	HLAunconditionalAttributeOwnership-Divestiture	HLAobjectInstance	HLAhandle	NA	HLAreliable	Receive
		HLAattributeList	HLAhandleList			
	HLAenableTimeRegulation	HLAlookahead	HLAtimeInterval	NA	HLAreliable	Receive
	HLAdisableTimeRegulation	NA	NA	NA	HLAreliable	Receive
	HLAenableTimeConstrained	NA	NA	NA	HLAreliable	Receive
	HLAdisableTimeConstrained	NA	NA	NA	HLAreliable	Receive
	HLAtimeAdvanceRequest	HLAtimeStamp	HLAlogicalTime	NA	HLAreliable	Receive
	HLAtimeAdvanceRequestAvailable	HLAtimeStamp	HLAlogicalTime	NA	HLAreliable	Receive
	HLAnextMessageRequest	HLAtimeStamp	HLAlogicalTime	NA	HLAreliable	Receive
	HLAnextMessageRequestAvailable	HLAtimeStamp	HLAlogicalTime	NA	HLAreliable	Receive
	HLAflushQueueRequest	HLAtimeStamp	HLAlogicalTime	NA	HLAreliable	Receive
	HLAenableAsynchronousDelivery	NA	NA	NA	HLAreliable	Receive
	HLAdisableAsynchronousDelivery	NA	NA	NA	HLAreliable	Receive
	HLAmodifyLookahead	HLAlookahead	HLAtimeInterval	NA	HLAreliable	Receive

Table 7—MOM parameter table *(continued)*

Interaction		Parameter	Datatype	Available dimensions	Transportation	Order
	HLAchangeAttributeOrderType	HLAobjectInstance	HLAhandle	NA	HLAreliable	Receive
		HLAattributeList	HLAhandleList			
		HLAsendOrder	HLAorderType			
	HLAchangeInteractionOrderType	HLAinteractionClass	HLAhandle	NA	HLAreliable	Receive
		HLAsendOrder	HLAorderType			
HLAmanager	HLAfederation	NA	NA	NA	HLAreliable	Receive
HLAmanager.HLAfederation	HLAadjust	NA	NA	NA	HLAreliable	Receive
HLAmanager.HLAfederation.HLAadjust	HLAsetSwitches	HLAautoProvide	HLAswitch	NA	HLAreliable	Receive
HLAmanager.HLAfederation.HLArequest	HLArequestSynchronizationPoints	NA	NA	NA	HLAreliable	Receive
	HLArequestSynchronizationPointStatus	HLAsyncPointName	HLAunicodeString	NA	HLAreliable	Receive
	HLArequestFOMmoduleData	HLAFOMmodule-Indicator	HLAindex	NA	HLAreliable	Receive
	HLArequestMIMData	NA	NA	NA	HLAreliable	Receive
HLAmanager.HLAfederation.HLAreport	HLAreportSynchronizationPoints	HLAsyncPoints	HLAsyncPointList	NA	HLAreliable	Receive
	HLAreportSynchronizationPointStatus	HLAsyncPointName	HLAunicodeString	NA	HLAreliable	Receive
		HLAsyncPointFeder-ates	HLAsyncPointFederateList			
	HLAreportFOMmoduleData	HLAFOMmoduleIn-dicator	HLAindex	NA	HLAreliable	Receive
		HLAFOMmoduleData	HLAunicodeString	NA	HLAreliable	Receive
	HLAreportMIMData	HLAMIMData	HLAunicodeString	NA	HLAreliable	Receive

Table 8—MOM dimension table

Name	Datatype	Dimension upper bound	Normalization function	Value when unspecified
HLAfederate	HLAnormalizedFederateHandle	RTI implementation dependent	*Normalize Federate Handle* service	Excluded
HLAserviceGroup	HLAnormalizedServiceGroup	7	*Normalize Service Group* service	Excluded

Table 9—MOM basic data representation table

Name	Size in bits	Interpretation	Endian	Encoding
HLAinteger32BE	32	Integer in the range $[-2^{31}, 2^{31}-1]$.	Big	32-bit twos-complement signed integer. The most significant bit contains the sign.
HLAoctetPairBE	16	16-bit value.	Big	Assumed to be portable among devices.
HLAoctet	8	8-bit value.	Big	Assumed to be portable among hardware devices.
NOTE—This table does not include the complete set of predefined basic data representations from the OMT specification (found in IEEE Std 1516.2-2010). Only the basic data representations needed for the MOM are present here.				

Table 10—MOM simple datatype table

Name	Representation	Units	Resolution	Accuracy	Semantics
HLAunicodeChar	HLAoctetPairBE	NA	NA	NA	Unicode UTF-16 character (see *The Unicode Standard*).
HLAbyte	HLAoctet	NA	NA	NA	Uninterpreted 8-bit byte.
HLAcount	HLAinteger32BE	NA	NA	NA	NA
HLAseconds	HLAinteger32BE	s	NA	NA	NA
HLAmsec	HLAinteger32BE	ms	NA	NA	NA
HLAnormalizedFederateHandle	HLAinteger32BE	NA	NA	NA	The normalized value of a federate handle as returned by the *Normalize Federate Handle* service. The value is appropriate for defining the range of the HLAfederate dimension for regions with this dimension.
HLAindex	HLAinteger32BE	NA	NA	NA	NA
NOTE—This table does not include the complete set of predefined simple datatypes from the OMT specification (found in IEEE Std 1516.2-2010). Only the simple datatypes needed for the MOM are present here.					

Table 11—MOM enumerated datatype table

Name	Representation	Enumerator	Values	Semantics
HLAboolean	HLAinteger32BE	HLAfalse	0	Standard boolean type
		HLAtrue	1	
HLAfederateState	HLAinteger32BE	ActiveFederate	1	State of the federate
		FederateSaveInProgress	3	
		FederateRestoreInProgress	5	
HLAtimeState	HLAinteger32BE	TimeGranted	0	State of time advancement
		TimeAdvancing	1	
HLAownership	HLAinteger32BE	Unowned	0	NA
		Owned	1	
HLAresignAction	HLAinteger32BE	DivestOwnership	1	Action to be performed by RTI in conjunction with resignation
		DeleteObjectInstances	2	
		CancelPendingAcquisitions	3	
		DeleteObjectInstancesThenDivestOwnership	4	
		CancelPendingAcquisitionsThenDelete-ObjectInstancesThenDivestOwnership	5	
		NoAction	6	
HLAorderType	HLAinteger32BE	Receive	0	Order type to be used for sending attributes or interactions
		TimeStamp	1	
HLAswitch	HLAinteger32BE	Enabled	1	NA
		Disabled	0	
HLAsyncPointStatus	HLAinteger32BE	NoActivity	0	Joined federate synchronization point status
		AttemptingToRegisterSyncPoint	1	
		MovingToSyncPoint	2	
		WaitingForRestOfFederation	3	

Table 11—MOM enumerated datatype table *(continued)*

Name	Representation	Enumerator	Values	Semantics
HLAnormalizedServiceGroup	HLAinteger32BE	FederationManagement	0	Service group identifier
		DeclarationManagement	1	
		ObjectManagement	2	
		OwnershipManagement	3	
		TimeManagement	4	
		DataDistributionManagement	5	
		SupportServices	6	

Table 12—MOM array datatype table

Name	Element Type	Cardinality	Encoding	Semantics
HLAunicodeString	HLAunicodeChar	Dynamic	HLAvariableArray	Unicode string representation.
HLAopaqueData	HLAbyte	Dynamic	HLAvariableArray	Uninterpreted sequence of bytes.
HLAhandle	HLAbyte	Dynamic	HLAvariableArray	Encoded value of a handle. The encoding is based on the type of handle.
HLAtransportationName	HLAunicodeChar	Dynamic	HLAvariableArray	String whose legal value shall be a name from any row in the OMT Transportation Table (see IEEE Std 1516.2-2010).
HLAupdateRateName	HLAunicodeChar	Dynamic	HLAvariableArray	String whose legal value shall be a name from any row in the OMT Update Rate Table (see IEEE Std 1516.2-2010.
HLAlogicalTime	HLAbyte	Dynamic	HLAvariableArray	An encoded logical time. An empty array shall indicate that the value is not defined.
HLAtimeInterval	HLAbyte	Dynamic	HLAvariableArray	An encoded logical time interval. An empty array shall indicate that the value is not defined.
HLAhandleList	HLAhandle	Dynamic	HLAvariableArray	List of encoded handles.
HLAinteractionSubList	HLAinteractionSubscription	Dynamic	HLAvariableArray	List of interaction subscription indicators.
NOTE—This table does not include the complete set of predefined array datatypes from the OMT specification (found in IEEE Std 1516.2-2010). Only the array datatypes needed for the MOM are present here.				

Table 12—MOM array datatype table *(continued)*

Name	Element Type	Cardinality	Encoding	Semantics
HLAargumentList	HLAunicodeString	Dynamic	HLAvariableArray	List of arguments.
HLAmoduleDesignatorList	HLAunicodeString	Dynamic	HLAvariableArray	List of designators of FOM modules
HLAobjectClassBasedCounts	HLAobjectClassBasedCount	Dynamic	HLAvariableArray	List of counts of various items based on object class. In all MOM interactions that have a parameter of datatype 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.
HLAinteractionCounts	HLAinteractionCount	Dynamic	HLAvariableArray	List of interaction counts. In all MOM interactions that have a parameter of datatype 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.
HLAsyncPointList	HLAunicodeString	Dynamic	HLAvariableArray	List of names of synchronization points.
HLAsyncPointFederateList	HLAsyncPointFederate	Dynamic	HLAvariableArray	List of joined federates and the synchronization status of each.

NOTE—This table does not include the complete set of predefined array datatypes from the OMT specification (found in IEEE Std 1516.2-2010). Only the array datatypes needed for the MOM are present here.

Table 13—MOM fixed record datatype table

Record name	Field			Encoding	Semantics
	Name	Type	Semantics		
HLAinteractionSubscription	HLAinteractionClass	HLAhandle	Encoded interaction class handle	HLAfixedRecord	Interaction subscription information.
	HLAactive	HLAboolean	Whether subscription is active (HLAtrue = active)		
HLAobjectClassBasedCount	HLAobjectClass	HLAhandle	Encoded object class handle	HLAfixedRecord	Object class and count of associated items.
	HLAcount	HLAcount	Number of items		
HLAinteractionCount	HLAinteractionClass	HLAhandle	Encoded interaction class handle	HLAfixedRecord	Count of interactions of a class.
	HLAinteractionCount	HLAcount	Number of interactions		
HLAsyncPointFederate	HLAfederate	HLAhandle	Encoded joined federate handle	HLAfixedRecord	A particular joined federate and its synchronization point status.
	HLAfederateSyncStatus	HLAsyncPointStatus	Synchronization status of the particular joined federate		

Table 14—MOM object class definitions table

Object	Semantics
HLAmanager	This object class is the root class of all MOM object classes.
HLAmanager.HLAfederation	This object class shall contain RTI state variables relating to a federation execution. The RTI shall publish it and shall register one object instance for the federation execution. The RTI shall respond to the invocation, by any federate, of the *Request Attribute Value Update* service for this object class or for any instance attribute of an object instance of this class by supplying values via the normal instance attribute update mechanism, regardless of whether the attribute has a datatype of static or conditional. In addition to its responsibility to update attributes of object instances of this class when those updates are explicitly requested, the RTI shall automatically update instance attributes of object instances of this class according to the update policy of the attribute, which is determined by the update type of the class attribute in Table 6. Attributes that have an update type of Conditional shall have update conditions as defined in the Table 6.
HLmanager.HLAfederate	This object class shall contain RTI state variables relating to a joined federate. The RTI shall publish it and shall register one object instance for each joined federate in a federation. Dynamic attributes that shall be contained in an object instance shall be updated periodically, where the period should be determined by an interaction of the class HLAmanager.HLAfederate.HLAadjust.HLAsetTiming. If this value is never set or is set to zero, no periodic update shall be performed by the RTI. The RTI shall respond to the invocation, by any federate, of the *Request Attribute Value Update* service for this object class or for any instance attribute of an object instance of this class by supplying values via the normal instance attribute update mechanism, regardless of whether the attribute has a datatype of static, periodic, or conditional. In addition to its responsibility to update attributes of object instances of this class when those updates are explicitly requested, the RTI shall automatically update instance attributes of object instances of this class according to the update policy of the attribute, which is determined by the update type of the class attribute in Table 6. For attributes that have an update type of Periodic, the update wall-clock time interval shall be determined by the HLAreportPeriod parameter in an interaction of classHLAmanager.HLAfederate.HLAadjust.HLAsetTiming. If this value is never set or is set to zero, no periodic updates shall be performed by the RTI. Attributes that have an update type of Conditional shall have update conditions as defined in the Table 6.

Table 15—MOM interaction class definitions table

Interaction		Semantics
HLAmanager		Root class of MOM interactions.
HLAmanager	HLAfederate	Root class of MOM interactions that deal with a specific joined federate.
HLAmanager.HLAfederate	HLAadjust	Permit a joined federate to adjust the RTI state variables associated with another joined federate.
	HLArequest	Permit a joined federate to request RTI data about another joined federate.
	HLAreport	Report RTI data about a joined federate. The RTI shall send these interactions in response to interactions of class HLAmanager.HLAfederate.HLArequest that correspond to services that are normally invoked by federates.
	HLAservice	The interaction class shall be acted upon by the RTI. These interactions shall invoke HLA services on behalf of another joined federate. They shall cause the RTI to react as if the service has been invoked by that other joined federate. If exceptions arise as a result of the use of these interactions, they shall be reported via the HLAmanager.HLAfederate.HLAreport.HLAreportMOMexception interaction to all joined federates that subscribe to this interaction. There are two ways an error can occur: The sending federate does not provide all the required arguments as parameters, or the preconditions of the spoofed service are not met. Each type of error is reported via HLAmanager.HLAfederate.HLAreport.HLAreportMOMexception. NOTE—These interactions shall have the potential to disrupt normal federation execution and should be used with great care.
HLAmanager.HLAfederate.HLAadjust	HLAsetTiming	Adjust the period between updates of the HLAmanager.HLAfederate object instance for the specified joined federate. If this interaction is never sent, the RTI shall not perform periodic updates.
	HLAmodifyAttributeState	Modify the ownership state of an attribute of an object instance for the specified joined federate. If the interaction is used to give ownership of the instance attribute to the specified joined federate and another joined federate currently owns the instance attribute, the owning joined federate shall be divested of ownership of the instance attribute before ownership is granted to the specified joined federate. No notification of change of ownership of the instance attribute shall be provided to either joined federate. In order for ownership of the instance attribute to be granted to the specified joined federate, the following conditions shall be true: – The specified joined federate knows about the object instance. – The specified joined federate is publishing the corresponding class attribute at the known class of the specified object instance at that joined federate. – The specified instance attribute is not owned by the RTI (i.e., it is not a predefined attribute of a MOM object class). If one or more of the above conditions are not met, the interaction shall have no effect, and an error shall be reported via an interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportMOMexception.
	HLAsetSwitches	Set the values of joined federate-specific switches. A joined federate may send individual declared parameters of this subclass.

Table 15—MOM interaction class definitions table *(continued)*

Interaction		Semantics
HLAmanager.HLAfederate.HLArequest	HLArequestPublications	Request that the RTI send report interactions that contain the publication data of a joined federate. It shall result in one interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportInteractionPublication and one interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportObjectClassPublication for each object class published. If the joined federate is published to no object classes, then one of class HLAmanager.HLAfederate.HLAreport.HLAreportObjectClassPublication shall be sent as a NULL response with the HLAobjectClassCount parameter having a value of 0.
	HLArequestSubscriptions	Request that the RTI send report interactions that contain the subscription data of a joined federate. 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. If the joined federate is subscribed to no object classes, then one interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportObjectClassSubscription shall be sent as a NULL response with the HLAobjectClassCount parameter having a value of 0.
	HLArequestObjectInstancesThatCanBeDeleted	Request that the RTI send a report interaction that contains the object instances that can be deleted at a joined federate. It shall result in one interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportObjectInstancesThatCanBeDeleted.
	HLArequestObjectInstancesUpdated	Request that the RTI send a report interaction that contains the object instance updating responsibility of a joined federate. It shall result in one interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportObjectInstancesUpdated.
	HLArequestObjectInstancesReflected	Request that the RTI send a report interaction that contains the object instances for which a joined federate has had a *Reflect Attribute Values* † service invocation. It shall result in one interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportObjectInstancesReflected.
	HLArequestUpdatesSent	Request that the RTI send a report interaction that contains the number of updates that a joined federate has generated. It shall result in one interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportUpdatesSent for each transportation type.
	HLArequestInteractionsSent	Request that the RTI send a report interaction that contains the number of interactions that a joined federate has generated. This count shall include interactions sent with region. It shall result in one interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportInteractionsSent for each transportation type.
	HLArequestReflectionsReceived	Request that the RTI send a report interaction that contains the number of reflections that a joined federate has received. It shall result in one interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportReflectionsReceived for each transportation type.
HLAmanager.HLAfederate.HLArequest	HLArequestInteractionsReceived	Request that the RTI send a report interaction that contains the number of interactions that a joined federate has received. It shall result in one interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportInteractionsReceived for each transportation type.
	HLArequestObjectInstanceInformation	Request that the RTI send a report interaction that contains the information that a joined federate maintains on a single object instance. It shall result in one interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportObjectInstanceInformation.

Table 15—MOM interaction class definitions table *(continued)*

Interaction		Semantics
	HLArequestFOMmoduleData	Request that the RTI shall send a report interaction with the content of the specified FOM module that was specified by the federate. The FOM module is indicated by the order number in the federate's HLAFOMmoduleDesignatorList attribute.
HLAmanager.HLAfederate.HLAreport	HLAreportObjectClassPublication	The interaction shall be sent by the RTI in response to an interaction of class HLAmanager.HLAfederate.HLArequest.HLArequestPublications. It shall report the attributes of one object class published by the joined federate. One of these interactions shall be sent for each object class containing attributes that are published by the joined federate. If the joined federate is published to no object classes, then a single interaction shall be sent as a NULL response with the HLAobjectClassCount parameter having a value of 0.
	HLAreportInteractionPublication	The interaction shall be sent by the RTI in response to an interaction of class HLAmanager.HLAfederate.HLAfequest.HLArequestPublications. It shall report the interaction classes published by the joined federate. If the joined federate is published to no interaction classes, then a single interaction shall be sent as a NULL response with the HLAinteractionClassList parameter having an undefined value (i.e., 0 length array).
	HLAreportObjectClassSubscription	The interaction shall be sent by the RTI in response to an interaction of class HLAmanager.HLAfederate.HLArequest.HLArequestSubscriptions. It shall report the attributes of one object class subscribed to by the joined federate. One of these interactions shall be sent for each combination of object class and passive subscription indicator subscribed to by the joined federate. This information shall reflect related DDM usage. If the joined federate has no subscribed object classes, then a single interaction shall be sent as a NULL response with the HLAnumberOfClasses parameter having a value of 0.
	HLAreportInteractionSubscription	The interaction shall be sent by the RTI in response to an interaction of class HLAmanager.HLAfederate.HLArequest.HLArequestSubscriptions. It shall report the interaction classes subscribed to by the joined federate. This information shall reflect related DDM usage. If the joined federate has no subscribed interaction classes, then a single interaction shall be sent as a NULL response with the HLAinteractionClassList parameter having an undefined value.
	HLAreportObjectInstancesThatCanBeDeleted	The interaction shall be sent by the RTI in response to an interaction of class HLAmanager.HLAfederate.HLArequest.HLArequestObjectInstancesThatCanBeDeleted. It shall report the number of object instances (by registered class of the object instances) whose **HLAprivilegeToDeleteObject** attributes are owned by the joined federate. If the joined federate has no objects that can be deleted, then a single interaction shall be sent with the HLAobjectInstanceCounts parameter having an undefined value.
	HLAreportObjectInstancesUpdated	The interaction shall be sent by the RTI in response to an interaction of class HLAmanager.HLAfederate.HLArequest.HLArequestObjectInstancesUpdated. It 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. If the joined federate has no object instances that are updated for a given transport type, then a single interaction shall be sent as a NULL response with the HLAobjectInstanceCounts parameter having an undefined value.

Table 15—MOM interaction class definitions table *(continued)*

Interaction		Semantics
HLAmanager.HLAfederate.HLAreport	HLAreportObjectInstancesReflected	The interaction shall be sent by the RTI in response to an interaction of class HLAmanager.HLAfederate.HLArequest.HLArequestObjectInstancesReflected. It shall report the number of object instances (by registered class of the object instances) for which the joined federate has had a *Reflect Attribute Values* † service invocation. If the joined federate has no object instances that are reflected for a given transport type, then a single interaction shall be sent as a NULL response with the HLAobjectInstanceCounts parameter having an undefined value.
	HLAreportUpdatesSent	The interaction shall be sent by the RTI in response to an interaction of class HLAmanager.HLAfederate.HLArequest.HLArequestUpdatesSent. It shall report the number of updates sent (by registered class of the object instances of the updates) by the joined federate since the beginning of the federation execution. One interaction of this class shall be sent by the RTI for each transportation type used. If the joined federate has no updates sent for a given transportation type, then a single interaction shall be sent as a NULL response with the HLAupdateCounts parameter having an undefined value.
	HLAreportReflectionsReceived	The interaction shall be sent by the RTI in response to an interaction of class HLAmanager.HLAfederate.HLArequest.HLArequestReflectionsReceived. It shall report the number of reflections received (by registered class of the object instances of the reflects) by the joined federate since the beginning of the federation execution. One interaction of this class shall be sent by the RTI for each transportation type used. If the joined federate has no reflections received for a given transportation type, then a single interaction shall be sent as a NULL response with the HLAreflectCounts parameter having an undefined value.
	HLAreportInteractionsSent	The interaction shall be sent by the RTI in response to an interaction of class HLAmanager.HLAfederate.HLArequest.HLArequestInteractionsSent. It shall report the number of interactions sent (by sent class of the interactions) by the joined federate since the beginning of the federation execution. This count shall include interactions sent with region. One interaction of this class shall be sent by the RTI for each transportation type used. If the joined federate has no interactions sent for a given transportation type, then a single interaction shall be sent as a NULL response with the HLAinteractionCounts parameter having an undefined value.
	HLAreportInteractionsReceived	The interaction shall be sent by the RTI in response to an interaction of class HLAmanager.HLAfederate.HLArequest.HLArequestInteractionsReceived. It shall report the number of interactions received (by sent class of the interactions) by the joined federate since the beginning of the federation execution. One interaction of this class shall be sent by the RTI for each transportation type used. If the joined federate has no interactions received for a given transportation type, then a single interaction shall be sent as a NULL response with the HLAinteractionCounts parameter having an undefined value.
	HLAreportObjectInstanceInformation	The interaction shall be sent by the RTI in response to an interaction of class HLAmanager.HLAfederate.HLArequest.HLArequestObjectInstanceInformation. It shall report on a single object instance and portray the instance attributes of that object instance that are owned by the joined federate, the registered class of the object instance, and the known class of the object instance at that joined federate. If the joined federate does not know the object instance, a single interaction shall be sent as a NULL response with the HLAownedInstanceAttributeList parameter having an undefined value.

Table 15—MOM interaction class definitions table *(continued)*

Interaction		Semantics
HLAmanager.HLAfederate.HLAreport	HLAreportException	The interaction shall be sent by the RTI when an exception occurs as the result of a service invocation at the indicated joined federate. This interaction shall be sent only if the HLAmanager.HLAfederate.HLAadjust.HLAexceptionReporting switch changing the HLAreportingState parameter sets the parameter to HLAtrue for the indicated joined federate.
	HLAreportServiceInvocation	This interaction shall be sent by the RTI whenever an HLA service is invoked, either by the indicated joined federate or by the RTI at the indicated joined federate, and the service reporting switch is enabled for the indicated joined federate. This interaction shall always contain the arguments supplied by the service invoker. If the service invocation was successful, the interaction also shall contain the value returned to the invoker (if the service returns a value); otherwise, the interaction also shall contain an indication of the exception that was raised to the invoker.
	HLAreportMOMexception	The interaction shall be sent by the RTI when one the following occurs: a MOM interaction without all the necessary parameters is sent or an interaction that imitates a federate's invocation of an HLA service is sent and not all of the service's preconditions are met.
	HLAreportFederateLost	The interaction shall be sent when a federate has been lost from the federation due to a fault.
	HLAreportFOMmoduleData	The interaction shall be sent by the RTI in response to an interaction of class HLAmanager.HLAfederate.HLArequest.HLArequestFOMmoduleData. It shall report the content of the specified FOM module for the federate.
HLAmanager.HLAfederate.HLAservice	HLAresignFederationExecution	Cause the RTI to react as if the *Resign Federation Execution* service has been invoked by the specified joined federate using the specified arguments (see 4.10). A joined federate shall be able to send this interaction anytime.
	HLAsynchronizationPointAchieved	Cause the RTI to react as if the *Synchronization Point Achieved* service has been invoked by the specified joined federate using the specified arguments (see 4.14).
	HLAfederateSaveBegun	Cause the RTI to react as if the *Federate Save Begun* service has been invoked by the specified joined federate using the specified arguments (see 4.18).
	HLAfederateSaveComplete	Cause the RTI to react as if the *Federate Save Complete* service has been invoked by the specified joined federate using the specified arguments (see 4.19).
	HLAfederateRestoreComplete	Cause the RTI to react as if the *Federate Restore Complete* service has been invoked by the specified joined federate using the specified arguments (see 4.28).
	HLApublishObjectClassAttributes	Cause the RTI to react as if the *Publish Object Class Attributes* service has been invoked by the specified joined federate using the specified arguments (see 5.2).
	HLAunpublishObjectClassAttributes	Cause the RTI to react as if the *Unpublish Object Class Attributes* service has been invoked by the specified joined federate using the specified arguments (see 5.3).
	HLApublishInteractionClass	Cause the RTI to react as if the *Publish Interaction Class* service has been invoked by the specified joined federate using the specified arguments (see 5.4).

Table 15—MOM interaction class definitions table *(continued)*

Interaction		Semantics
	HLAunpublishInteractionClass	Cause the RTI to react as if the *Unpublish Interaction Class* service has been invoked by the specified joined federate using the specified arguments (see 5.5).
	HLAsubscribeObjectClassAttributes	Cause the RTI to react as if the *Subscribe Object Class Attributes* service has been invoked by the specified joined federate using the specified arguments (see 5.6).
	HLAunsubscribeObjectClassAttributes	Cause the RTI to react as if the *Unsubscribe Object Class Attributes* service has been invoked by the specified joined federate using the specified arguments (see 5.7).
	HLAsubscribeInteractionClass	Cause the RTI to react as if the *Subscribe Interaction Class* service has been invoked by the specified joined federate using the specified arguments (see 5.8).
	HLAunsubscribeInteractionClass	Cause the RTI to react as if the *Unsubscribe Interaction Class* service has been invoked by the specified joined federate using the specified arguments (see 5.9).
	HLAdeleteObjectInstance	Cause the RTI to react as if the *Delete Object Instance* service has been invoked by the specified joined federate using the specified arguments (see 6.14). If applicable, the retraction handle returned by invoking this service is discarded, and the scheduled action cannot be retracted.
	HLAlocalDeleteObjectInstance	Cause the RTI to react as if the *Local Delete Object Instance* service has been invoked by the specified joined federate using the specified arguments (see 6.16).
	HLArequestAttributeTransportation Type change	Cause the RTI to react as if the *Request Attribute Transportation Type Change* service has been invoked by the specified joined federate using the specified arguments (see 6.23).
	HLArequestInteractionTransportationTypeChange	Cause the RTI to react as if the *Request Interaction Transportation Type Change* service has been invoked by the specified joined federate using the specified arguments (see 6.27)
	HLAunconditionalAttributeOwnershipDivestiture	Cause the RTI to react as if the *Unconditional Attribute Ownership Divestiture* service has been invoked by the specified joined federate using the specified arguments (see 7.2).
	HLAenableTimeRegulation	Cause the RTI to react as if the *Enable Time Regulation* service has been invoked by the specified joined federate using the specified arguments (see 8.2).
	HLAdisableTimeRegulation	Cause the RTI to react as if the *Disable Time Regulation* service has been invoked by the specified joined federate using the specified arguments (see 8.4).
	HLAenableTimeConstrained	Cause the RTI to react as if the *Enable Time Constrained* service has been invoked by the specified joined federate using the specified arguments (see 8.5).
	HLAdisableTimeConstrained	Cause the RTI to react as if the *Disable Time Constrained* service has been invoked by the specified joined federate using the specified arguments (see 8.7).
HLAmanager.HLAfederate.HLAservice	HLAtimeAdvanceRequest	Cause the RTI to react as if the *Time Advance Request* service has been invoked by the specified joined federate using the specified arguments (see 8.8).
	HLAtimeAdvanceRequestAvailable	Cause the RTI to react as if the *Time Advance Request Available* service has been invoked by the specified joined federate using the specified arguments (see 8.9)

Table 15—MOM interaction class definitions table *(continued)*

Interaction		Semantics
	HLAnextMessageRequest	Cause the RTI to react as if the *Next Message Request* service has been invoked by the specified joined federate using the specified arguments (see 8.10)
	HLAnextMessageRequestAvailable	Cause the RTI to react as if the *Next Message Request Available* service has been invoked by the specified joined federate using the specified arguments (see 8.11)
	HLAflushQueueRequest	Cause the RTI to react as if the *Flush Queue Request* service has been invoked by the specified joined federate using the specified arguments (see 8.12)
	HLAenableAsynchronousDelivery	Cause the RTI to react as if the *Enable Asynchronous Delivery* service has been invoked by the specified joined federate using the specified arguments (see 8.14)
	HLAdisableAsynchronousDelivery	Cause the RTI to react as if the *Disable Asynchronous Delivery* service has been invoked by the specified joined federate using the specified arguments (see 8.15)
	HLAmodifyLookahead	Cause the RTI to react as if the *Modify Lookahead* service has been invoked by the specified joined federate using the specified arguments (see 8.19)
	HLAchangeAttributeOrderType	Cause the RTI to react as if the *Change Attribute Order Type* service has been invoked by the specified joined federate using the specified arguments (see 8.23)
	HLAchangeInteractionOrderType	Cause the RTI to react as if the *Change Interaction Order Type* service has been invoked by the specified joined federate using the specified arguments (see 8.24)
HLAmanager	HLAfederation	Root class of MOM interactions that deal with a specific federation execution.
HLAmanager.HLAfederation	HLAadjust	Permit a joined federate to adjust the RTI state variables associated with a federation execution.
	HLArequest	Permit a federate to request RTI data about a specific federation execution.
	HLAreport	Permit a federate to receive RTI data about a specific federation execution.
HLAmanager.HLAfederation.HLAadjust	HLAsetSwitches	Set the values of federation execution-wide switches. A joined federate may send individual declared parameters of this subclass
HLAmanager.HLAfederation.HLArequest	HLArequestSynchronizationPoints	Request that the RTI send a report interaction that contains a list of all in-progress federation synchronization points. It shall result in one interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportSynchronizationPoints.
	HLArequestSynchronizationPointStatus	Request that the RTI send a report interaction that contains a list that includes each federate (and its synchronization status) that is associated with a particular synchronization point. It shall result in one interaction of class HLAmanager.HLAfederate.HLAreport.HLAreportSynchronizationPointStatus.
	HLArequestFOMmoduleData	Requests that the RTI shall send a report interaction with the content of the specified FOM module for the federation. The FOM module is indicated by the order number in the federation's HLAFOMmoduleDesignatorList attribute.
	HLArequestMIMData	Requests that the RTI shall send a report interaction with the content of the MIM for the federation.

Table 15—MOM interaction class definitions table *(continued)*

Interaction		Semantics
HLAmanager.HLAfederation.HLAreport	HLAreportSynchronizationPoints	The interaction shall be sent by the RTI in response to an interaction of class HLAmanager.HLAfederation.HLArequest.HLArequestSynchronizationPoints. It shall report the list of active synchronization points in the federation execution. If there are no active synchronization points in the federation execution, a single interaction shall be sent as a NULL response with the HLAsyncPoints parameter having an undefined value.
	HLAreportSynchronizationPointStatus	The interaction shall be sent by the RTI in response to an interaction of class HLAmanager.HLAfederation.HLArequest.HLArequestSynchronizationPointStatus. It shall report the status of a particular synchronization point. This report shall be a list that includes each federate (and its synchronization status) that is associated with the particular synchronization point. One interaction of class HLAmanager.HLAfederation.HLAreport.HLAreportSynchronizationPointStatus shall be sent by the RTI for each active synchronization point in the federation execution. If there is no active synchronization point with the given name from the request, then an interaction shall be sent as a NULL response with the HLAsyncPointFederates parameter having an undefined value.
	HLAreportFOMmoduleData	The interaction shall be sent by the RTI in response to an interaction of class HLAmanager.HLAfederation.HLArequest.HLArequestFOMmoduleData. It shall report the content of the specified FOM module for the federation.
	HLAreportMIMData	The interaction shall be sent by the RTI in response to an interaction of class HLAmanager.HLAfederation.HLArequest.HLArequest MIMData. It shall report the content of the MIM for the federation.

Table 16—MOM attribute definitions table

Class	Attribute	Semantics
HLAmanager.HLAfederate	HLAfederateHandle	Handle of the joined federate returned by a *Join Federation Execution* service invocation.
	HLAfederateName	Name of the joined federate supplied to a successful *Join Federation Execution* service invocation.
	HLAfederateType	Type of the joined federate specified by the joined federate when it joined the federation.
	HLAfederateHost	Host name of the computer on which the joined federate is executing.
	HLARTIversion	Version of the RTI software being used.
	HLAFOMmoduleDesignatorList	FOM module designators as specified by the federate when the *Join Federation Execution* service was invoked. Only FOM modules that added information to the FDD when they were loaded shall be added to the list.
	HLAtimeConstrained	Whether the time advancement of the joined federate is constrained by other joined federates.
	HLAtimeRegulating	Whether the joined federate influences the time advancement of other joined federates.

Table 16—MOM attribute definitions table *(continued)*

Class	Attribute	Semantics
	HLAasynchronousDelivery	Whether the RTI shall deliver RO messages to the joined federate while the joined federate's time manager state is not "Time Advancing" (only matters if the joined federate is time-constrained).
	HLAfederateState	State of the joined federate. The MOM may, but is not required to, update any HLAfederateState instance attribute values during the interval after the last federate in the federation execution invokes the *Federate Restore Complete* service but before the last *Federation Restored †* callback is invoked at some federate for a given federation restoration.
	HLAtimeManagerState	State of the joined federate's time manager.
	HLAlogicalTime	Joined federate's logical time. The initial value of this information is the initial value of federation time of the TRADT.
	HLAlookahead	Minimum duration into the future that a TSO message will be scheduled. The value shall not be defined if the joined federate is not time-regulating.
	HLAGALT	Joined federate's GALT. The value shall not be defined if GALT is not defined for the joined federate.
	HLALITS	Joined federate's LITS. The value shall not be defined if LITS is not defined for the joined federate.
	HLAROlength	Number of RO messages queued for delivery to the joined federate.
	HLATSOlength	Number of TSO messages queued for delivery to the joined federate.
	HLAreflectionsReceived	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 that have been received at the joined federate).
	HLAupdatesSent	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).
HLAmanager.HLAfederate	HLAinteractionsReceived	Total number of interactions received by the joined federate.
	HLAinteractionsSent	Total number of interactions sent by the joined federate. This information shall reflect related DDM usage.
	HLAobjectInstancesThatCanBedeleted	Total number of object instances whose **HLAprivilegeToDeleteObject** attribute is owned by the joined federate.
	HLAobjectInstancesUpdated	Total number of object instances for which the joined federate has successfully invoked the *Update Attribute Values* service.
	HLAobjectInstancesReflected	Total number of object instances for which the joined federate has had a *Reflect Attribute Values †* service invocation.
	HLAobjectInstancesDeleted	Total number of times the *Delete Object Instance* service was successfully invoked by the joined federate since the federate joined the federation.
	HLAobjectInstancesRemoved	Total number of times the *Remove Object Instance †* service was invoked for the joined federate since the federate joined the federation.
	HLAobjectInstancesRegistered	Total number of times the *Register Object Instance* service and *Register Object Instance with Regions* service were successfully invoked by the joined federate since the federate joined the federation.
	HLAobjectInstancesDiscovered	Total number of times the *Discover Object Instance †* service was successfully invoked for the joined federate since the federate joined the federation. 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.

Table 16—MOM attribute definitions table *(continued)*

Class	Attribute	Semantics
	HLAtimeGrantedTime	Wall-clock time duration that the federate has spent in the Time Granted state since the last update of this attribute. When the HLAtimeGrantedTime and the HLAtimeAdvancingTime attributes are initially updated, their values shall be the wall-clock time duration that the federate has spent in the state since the federate has been joined to the federation execution.
	HLAtimeAdvancingTime	Wall-clock time duration that the federate has spent in the Time Advancing state since the last update of this attribute. When the HLAtimeGrantedTime and the HLAtimeAdvancingTime attributes are initially updated, their values shall be the wall-clock time duration that the federate has spent in the state since the federate has been joined to the federation execution.
	HLAconveyRegionDesignatorSets	Value of the joined federate's Convey Region Designator Sets Switch. Updated when value of switch changes.
	HLAconveyProducingFederate	Value of the joined federate's Convey Producing Federate Switch. Updated when value of switch changes.
HLAmanager.HLAfederation	HLAfederationName	Name of the federation to which the joined federate belongs.
	HLAfederatesInFederation	Identifiers of joined federates that are joined to the federation.
	HLARTIversion	Version of the RTI software.
	HLAMIMDesignator	Designator associated with the MIM specified in the *Create Federation Execution* service invocation. In case the RTI has supplied the standard MIM, the designator shall be "HLAstandardMIM."
	HLAFOMmoduleDesignatorList	FOM module designators for the federation as specified in the *Create Federation Execution* service and *Join Federation Execution* service invocations. Only FOM modules that added information to the FDD when they were loaded shall be added to the list.
	HLAcurrentFDD	The current FDD realized as a result of preceding successful *Create Federation Execution* and *Join Federation Execution* service invocations.
	HLAtimeImplementationName	Name of the time implementation as supplied to the *Create Federation Execution* service when the federation was created.
	HLAlastSaveName	Name associated with the last federation state save (null if no saves have occurred).
	HLAlastSaveTime	Logical time at which the last federation state save occurred. If the last save was not a timed save, then the HLAlastSaveTime attribute value shall be an empty (zero-length) HLAlogicalTime array to indicate that the value of the HLAlastSaveTime attribute is undefined. If no timed saves have occurred, the value shall be an empty (zero-length) HLAlogicalTime array.
	HLAnextSaveName	Name associated with the next federation state save (null if no saves are scheduled).
	HLAnextSaveTime	Logical time at which the next federation state timed save is scheduled. If no timed saves are scheduled, the value shall be an empty (zero-length) HLAlogicalTime array.
	HLAautoProvide	Value of federation-wide Auto-Provide Switch. Updated when value of switch changes.

Table 17—MOM parameter definitions table

Class		Parameter	Semantics
HLAmanager	HLAfederate	HLAfederate	Handle of the joined federate that was provided when joining.
HLAmanager.HLAfederate.HLAadjust	HLAsetTiming	HLAreportPeriod	Number of seconds between updates of instance attribute values of the HLAfederate object instance. (A zero value causes periodic updates to cease.) If no interaction of class HLAmanager.HLAfederate.HLAadjust.HLAsetTiming has been sent, then no periodic updates of MOM attribute values shall be generated.
	HLAmodifyAttributeState	HLAobjectInstance	Handle of the object instance whose attribute state is being changed.
		HLAattribute	Handle of the instance attribute whose state is being changed.
		HLAattributeState	New state for the attribute of the object instance.
	HLAsetSwitches	HLAconveyRegionDesignatorSets	Set the joined federate's Convey Region Designator Sets Switch to the provided value.
		HLAconveyProducingFederate	Set the joined federate's Convey Producing Federate Switch to the provided value.
		HLAserviceReporting	Set the joined federate's service reporting switch to the provided value. Specify whether to report service invocations to or from the specified joined federate via HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation interactions. If the specified joined federate is subscribed to the HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation interaction, all attempts to enable for that joined federate by enabling this switch shall fail and be reported via the normal MOM interaction failure means. (default = disabled).
		HLAexceptionReporting	Set the joined federate's exception reporting switch to the provided value. Specify whether the RTI shall report service invocation exceptions via HLAmanager.HLAfederate.HLAreport.HLAreportException interactions (default = disabled).
HLAmanager.HLAfederate.HLArequest	HLArequestObjectInstanceInformation	HLAobjectInstance	Handle of the object instance for which information is being requested.
	HLArequestFOMmoduleData	HLAFOMmoduleIndicator	Indicates order number of requested FOM module.

Table 17—MOM parameter definitions table *(continued)*

Class		Parameter	Semantics
HLAmanager.HLAfederate.HLAreport	HLAreportObjectClassPublication	HLAnumberOfClasses	The number of object classes for which the joined federate publishes attributes. This parameter shall be 0 in a NULL response.
		HLAobjectClass	The object class whose publication is being reported. This parameter shall be omitted in a NULL response.
		HLAattributeList	List of handles of HLAobjectClass attributes that the joined federate is publishing. This parameter shall be omitted in a NULL response.
	HLAreportInteractionPublication	HLAinteractionClassList	List of interaction classes that the joined federate is publishing. This parameter shall be an empty list in a NULL response.
	HLAreportObjectClassSubscription	HLAnumberOfClasses	The number of object class and passive indicator combinations for which the joined federate subscribes to attributes. This information shall reflect related DDM usage. This parameter shall be 0 in a NULL response.
		HLAobjectClass	The object class whose subscription is being reported. This parameter shall be omitted in a NULL response.
		HLAactive	Whether the subscription is active. This parameter shall be omitted in a NULL response.
		HLAmaxUpdateRate	Name of the maximum subscribed update rate. This parameter shall be omitted in a NULL response.
		HLAattributeList	List of handles of class attributes to which the joined federate is subscribing. This parameter shall be omitted in a NULL response.
	HLAreportInteractionSubscription	HLAinteractionClassList	List of interaction class and subscription type pairs. Each pair consists of the handle of an interaction class that the joined federate is subscribed to and whether the joined federate is actively subscribing. This information shall reflect related DDM usage. This parameter shall be an empty list in a NULL response.
	HLAreportObjectInstancesThatCanBeDeleted	HLAobjectInstanceCounts	A list of object instance counts. Each object instance count consists of an object class handle and the number of object instances of that class. This parameter shall be an empty list in a NULL response.

Table 17—MOM parameter definitions table *(continued)*

Class		Parameter	Semantics
HLAmanager.HLAfederate.HLAreport	HLAreportObjectInstancesUpdated	HLAobjectInstanceCounts	List of object instance counts. Each object instance count consists of an object class handle and the number of object instances of that class. This parameter shall be an empty list in a NULL response.
	HLAreportobjectInstancesReflected	HLAobjectInstanceCounts	List of object instance counts. Each object instance count consists of an object class handle and the number of object instances of that class. This parameter shall be an empty list in a NULL response.
	HLAreportUpdatesSent	HLAtransportation	Transportation type used in sending updates.
		HLAupdateCounts	List of update counts. Each update count consists of an object class handle and the number of instance attribute updates sent of that class. 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 a HLAmanager.HLAfederate.HLAreport.HLAreportUpdatesSent 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 HLAreportUpdatesSent interaction that is sent for that transportation type shall have an empty HLAobjectClassBasedCount array. 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 shall be in the HLAobjectClassBasedCount array of the HLAmanager.HLAfederate.HLAreport.HLAreportUpdatesSent interaction for that transportation type. This parameter shall be an empty list in a NULL response for the given transportation type.

Table 17—MOM parameter definitions table *(continued)*

Class		Parameter	Semantics
HLAmanager.HLAfederate.HLAreport	HLAreportReflectionsReceived	HLAtransportation	Transportation type used in receiving reflections.
		HLAreflectCounts	List of reflection counts. Each reflection count consists of an object class handle and the number of instance attribute reflections received of that class. 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 a 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. 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 parameter shall be an empty list in a NULL response for the given transportation type.
	HLAreportInteractionsSent	HLAtransportation	Transportation type used in sending interactions.
		HLAinteractionCounts	List of interaction counts. Each interaction count consists of an interaction class handle and the number of interactions of that class. This information shall reflect related DDM usage. This parameter shall be an empty list in a NULL response for the given transportation type.
HLAmanager.HLAfederate.HLAreport	HLAreportInteractionsReceived	HLAtransportation	Transportation type used in receiving interactions.
		HLAinteractionCounts	List of interaction counts. Each interaction count consists of an interaction class handle and the number of interactions of that class. This parameter shall be an empty list in a NULL response for the given transportation type.

Table 17—MOM parameter definitions table *(continued)*

Class		Parameter	Semantics
	HLAreportobjectInstanceInformation	HLAobjectInstance	Handle of the object instance for which the interaction was sent.
		HLAownedInstanceAttributeList	List of the handles of all instance attributes, of the object instance, owned by the joined federate. This parameter shall be an empty list in a NULL response.
		HLAregisteredClass	Handle of the registered class of the object instance. This parameter shall be omitted in a NULL response.
		HLAknownClass	Handle of the known class of the object instance at the joined federate. This parameter shall be omitted in a NULL response.
	HLAreportException	HLAservice	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 preconditions 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. The name of the interaction class provided shall always be fully qualified, as defined in the OMT, so as to avoid potential ambiguities.
		HLAexception	Textual depiction of the exception.
HLAmanager.HLAfederate.HLAreport	HLAreportServiceInvocation	HLAservice	Textual name of the service.
		HLAsuccessIndicator	Whether the service invocation was successful. Exception values are returned along with a HLAfalse value.
		HLAsuppliedArguments	Textual depiction of the arguments supplied in the service invocation.
		HLAreturnedArguments	Textual depiction of the argument returned by the service invocation. The list is null if the service does not normally return a value or if HLAsuccessIndicator is HLAfalse. Each returned argument is an element on the list. The number of returned arguments depends upon the service narrative and not any particular API and services may have more than one returned argument.
		HLAexception	Textual description of the exception raised by this service invocation (null if *i* is HLAtrue).

Table 17—MOM parameter definitions table *(continued)*

Class		Parameter	Semantics
		HLAserialNumber	This is a per-joined federate serial number that shall start at zero and shall increment by 1 for each HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation interaction that represents service invocations to or from the respective joined federate.
	HLAreportMOMexception	HLAservice	Name of the service interaction that had a problem or raised an exception.
		HLAexception	Textual depiction of the problem/exception.
		HLAparameterError	HLAtrue if there was an incorrect number of interaction parameters or a parameter was incorrectly formatted. HLAfalse otherwise.
	HLAreportFederateLost	HLAfederateName	Name of the lost joined federate
		HLAtimestamp	Last known good timestamp to which the lost joined federate was granted
		HLAfaultDescription	Textual description of the fault
	HLAreportFOMmoduleData	HLAFOMmoduleIndicator	Indicates order number of reported FOM module.
		HLAFOMmoduleData	Contents of the reported FOM module.
HLAmanager.HLAfederate.HLAservice	HLAresignFederationExecution	HLAresignAction	Action that the RTI is to take in conjunction with the resignation.
	HLAsynchronizationPointAchieved	HLAlabel	Label associated with the synchronization point.
	HLAfederateSaveComplete	HLAsuccessIndicator	Whether the federate save was successful.
	HLAfederateRestoreComplete	HLAsuccessIndicator	Whether the restore was successful.
	HLApublishObjectClassAttributes	HLAobjectClass	Object class for which the joined federate's publication shall change.
		HLAattributeList	List of handles of attributes of HLAobjectClass that the joined federate shall now publish.
	HLAunpublishObjectClassAttributes	HLAobjectClass	Object class for which the joined federate's unpublication shall change.
		HLAattributeList	List of handles of attributes of HLAobjectClass that the joined federate shall now unpublish.
	HLApublishInteractionClass	HLAinteractionClass	Interaction class that the joined federate shall publish.
	HLAunpublishInteractionClass	HLAinteractionClass	Interaction class that the joined federate shall no longer publish.

Table 17—MOM parameter definitions table *(continued)*

Class		Parameter	Semantics
HLAmanager.HLAfederate.HLAservice	HLAsubscribeObjectClassAttributes	HLAobjectClass	Object class for which the joined federate's subscription shall change.
		HLAattributeList	List of handles of attributes of HLAobjectClass to which the joined federate shall now subscribe.
		HLAactive	Whether the subscription is active.
	HLAunsubscribeObjectClassAttributes	HLAobjectClass	Object class for which the joined federate's subscription shall change.
		HLAattributeList	List of handles of attributes of HLAobjectClass to which the joined federate shall now unsubscribe.
	HLAsubscribeInteractionClass	HLAinteractionClass	Interaction class to which the joined federate shall subscribe.
		HLAactive	Whether the subscription is active.
	HLAunsubscribeInteractionClass	HLAinteractionClass	Interaction class to which the joined federate will no longer be subscribed.
	HLAdeleteObjectInstance	HLAobjectInstance	Handle of the object instance that is to be deleted.
		HLAtag	Tag associated with the deletion.
		HLAtimeStamp	Timestamp of the deletion (optional).
	HLAlocalDeleteObjectInstance	HLAobjectInstance	Handle of the object instance that is to be deleted.
	HLArequestAttributeTransportationTypeChange	HLAobjectInstance	Handle of the object instance whose attribute transportation type is to be changed.
		HLAattributeList	List of the handles of instance attributes whose transportation type is to be changed.
		HLAtransportation	Transportation type to be used for updating instance attributes in the list.
	HLArequestInteractionTransportationTypeChange	HLAinteractionClass	Interaction class whose transportation type is changed by this service invocation.
		HLAtransportation	Transportation type to be used for sending the interaction class.
HLAmanager.HLAfederate.HLAservice	HLAunconditionalAttributeOwnershipDivestiture	HLAobjectInstance	Handle of the object instance whose attributes' ownership is to be divested.
		HLAattributeList	List of handles of instance attributes belonging to *HLAobjectInstance* whose ownership is to be divested by the joined federate.
	HLAenableTimeRegulation	HLAlookahead	Lookahead to be used by the joined federate while regulating other joined federates.

Table 17—MOM parameter definitions table *(continued)*

Class		Parameter	Semantics
	HLAtimeAdvanceRequest	HLAtimeStamp	Timestamp requested.
	HLAtimeAdvanceRequestAvailable	HLAtimeStamp	Timestamp requested.
	HLAnextMessageRequest	HLAtimeStamp	Timestamp requested.
	HLAnextMessageRequestAvailable	HLAtimeStamp	Timestamp requested.
	HLAflushQueueRequest	HLAtimeStamp	Timestamp requested.
	HLAmodifyLookahead	HLAlookahead	New value for lookahead.
	HLAchangeAttributeOrderType	HLAobjectInstance	Handle of the object instance whose attribute order type is to be changed.
		HLAattributeList	List of the handles of instance attributes whose order type is to be changed.
		HLAsendOrder	Order type to be used for sending the instance attribute list.
	HLAchangeInteractionOrderType	HLAinteractionClass	Interaction class whose order type is changed by this service invocation.
		HLAsendOrder	Order type to be used for sending the interaction class.
HLAmanager.HLAfederation.HLAadjust	HLAsetSwitches	HLAautoProvide	Set the federation-wide Auto-Provide Switch to the provided value.
HLAmanager.HLAfederation.HLArequest	HLArequestSynchronizationPointStatus	HLAsyncPointName	Name of a particular synchronization point.
	HLArequestFOMmoduleData	HLAFOMmoduleIndicator	Indicates order number of requested FOM module.
HLAmanager.HLAfederation.HLAreport	HLAreportSynchronizationPoints	HLAsyncPoints	List of the in-progress federation execution synchronization points.
	HLAreportSynchronizationPointStatus	HLAsyncPointName	Name of a particular synchronization point.
		HLAsyncPointFederates	List of each federate (and its synchronization status) associated with the particular synchronization point.
	HLAreportFOMmoduleData	HLAFOMmoduleIndicator	Indicates order number of reported FOM module.
		HLAFOMmoduleData	Contents of the reported FOM module.
	HLAreportMIMData	HLAMIMData	Contents of the reported MIM.

12. Programming language mappings

12.1 Overview

The services described in this document to this point have been described in a programming-language-neutral way. This approach was taken because support is provided for multiple programming languages as described in the annexes to this document. This clause (i.e., Clause 12) provides the reader with a mapping from the abstract view from the body of this document (Clause 4 through Clause 11) to the programming-language-specific views in the APIs (Annex B through Annex D). It shall be transparent to the federation and participating federates which API or implementation language a specific federate uses.

The mappings described in this clause are

— Of designators from Clause 4 through Clause 10 to handles and names in the APIs
— From services in Clause 4 through Clause 10 to methods in the APIs
— From abstract datatypes in Clause 4 through Clause 10 to classes or datatypes in the APIs

12.2 Designators

As explained in 1.4.3, designators are a generalization of both names and handles. In most cases, what is referred to as a designator in Clause 4 through Clause 10 has both a name and a handle. Which specific form of a designator is used in the APIs is dependent on the language and service in which it is used. Examine the APIs to see which form is needed, although in most cases, the handle is used. Clause 10 presents services that shall be used to map names to handles and vice versa.

Names in most cases (e.g., object class names, attribute names, interaction class names) come from the FDD. As such, 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. As such, these names often are not predictable.

Handles in all cases are generated by the RTI. In most cases, where the handles correspond to names from the FDD, these handles shall be generated in a repeatable manner. In other words, 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 so that handles, rather than just the names, can be saved by federates. This property does not, however, imply that an RTI 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. In other words, the federate registering an object instance shall know it by the same handle as another federate that discovers the object instance.

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 informing it that one particular object instance is of particular importance. 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.

To send a handle as an attribute value or as a parameter value, the handle shall be encoded prior to sending, and it shall be reconstituted by the receiver. The precise means of encoding and reconstituting a handle is described for each API in 12.11.5.4, 12.12.6.4, and 12.13.4. However, a handle encoded using any API (e.g., C++) shall be reconstitutable in all APIs. Also, a handle of a given type (e.g., an object instance handle) shall be reconstitutable only 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 for each API.

12.3 Logical time, timestamps, and lookahead

In order to support customized implementations of logical time, timestamps, and lookahead, the C++ and Java APIs define a group of closely related abstract interfaces: **`LogicalTime`**, **`LogicalTimeInterval`**, and **`LogicalTimeFactory`**. These abstract interfaces have been designed to be implemented by the federate developer for use by an RTI implementation. This clause defines the general characteristics of the abstract interfaces, and each API annex presents API-specific details.

The **`LogicalTime`** abstract interface is used to represent logical time and timestamp values. It has methods to compare the relative order of one instance with another and to compare an instance with the initial value (e.g., zero) and with the final value (e.g., infinity). This generality is provided to allow federate programmers to define arbitrary lower and upper bounds on their implementation of the **`LogicalTime`** class. Methods shall exist to add and subtract a value to and from of an instance of **`LogicalTime`**. However, it is noted that the argument to these methods is of type **`LogicalTimeInterval`**, and not of type **`LogicalTime`**.

The **`LogicalTimeInterval`** abstract interface is similar to the **`LogicalTime`** abstract interface, but it is used to represent lookahead values. The chief differences in its interface are that there is no concept of a "final" or "largest possible" **`LogicalTimeInterval`** instance; instead, there are methods to test if a **`LogicalTimeInterval`** instance is zero or epsilon. Epsilon is the smallest nonzero **`LogicalTimeInterval`** that is possible with the particular implementation. This value is used by an RTI implementation to support time management services for federates with zero lookahead. The value of epsilon is intended to be constant. Methods shall exist to add and subtract a value to and from a **`LogicalTimeInterval`** instance.

`LogicalTime` and **`LogicalTimeInterval`** abstract interfaces shall provide methods to obtain a string representation suitable for printing. The set of abstract interfaces for **`LogicalTime`**, **`LogicalTimeInterval`**, and **`LogicalTimeFactory`** shall also support some way to

— Encode an instance of **`LogicalTime`** or **`LogicalTimeInterval`** into a compact, opaque representation suitable for network transmission into the provided variable array at the specified offset
— Create a **`LogicalTime`** or **`LogicalTimeInterval`** from the network representation in the provided variable array at the indicated offset.

The **`LogicalTimeFactory`** abstract interface shall have a method to create a **`LogicalTime`** instance whose value is the user-defined initial value and a method to create a **`LogicalTime`** instance whose value is the user-defined final value. The **`LogicalTimeFactory`** abstract interface shall have a method to create a **`LogicalTimeInterval`** instance whose value is zero and a method to create a **`LogicalTimeInterval`** instance whose value is the epsilon value.

To allow the users to get the **`LogicalTimeFactory`** from the RTI, the C++ and Java APIs add a getTimeFactory() method to the RTI ambassador interface. Each API returns a value appropriate for the language.

Implementers of the C++ and Java APIs shall provide, as part of their implementation, two implementations of the logical time abstract interfaces. One logical time abstract interface, the **`HLAInteger64Time`**, shall be based on the **`HLAInteger64BE`** type and the other, the **`HLAFloat64Time`**, shall be based on the **`HLAFloat64BE`** type. The following are intended to describe requirements on ALL time implementations.

Given time *T* and nonzero interval *I*, zero interval *Iz*, and epsilon interval *Ie*, all implementations of logical time shall ensure that the following statements are true:

T' = *T* + *I* => *T* < *T'*
T' = *T* – *I* => *T* > *T'*
T' = *T* + *Iz* => *T* == *T'*
T' = *T* – *Iz* => *T* == *T'*
*T*1 != *T*2 => |*T*1 – *T*2| >= *Ie*
*T*1 == *T*2 => *T*2 – *T*1 == *Iz*
I' = *I* + *Iz* => *I* == *I'*
I' = *I* – *Iz* => *I* == *I'*
*I*2 != *I*1 => |*I*2 – *I*1| >= *Ie*
*I*2 == *I*1 => *I*2 – *I*1 == *Iz*
I != *Iz* => *I* >= *Ie*
T > *T'* => *T'* < *T*

12.4 Standardized time types

An RTI shall support the standardized time representation shown in Table 18.

Table 18—Standardized time types

Property	64-bit integer time	64-bit float time
Symbolic name	"HLAinteger64Time"	"HLAfloat64Time"
Representation	64-bit integer	64-bit float
Encoded format	HLAinteger64BE	HLAfloat64BE
Initial value	0	0.0
Final value	2^{63} - 1	0x1.FFFFFFFFFFFFFP+1023
Zero value	0	0.0
Epsilon value	1	0x0.0000000000001P-1022

When a federate execution is created, the time representation used is specified using the symbolic name above. All time representations shall provide methods through which a time value can be increased and decreased by epsilon while guaranteeing that the increased value differs from the original value. The epsilon value for HLAinteger64Time shall equal 1.

12.4.1 HLAInteger64Time

The implementation of **`HLAInteger64Time`** shall provide the following:

— An implementation of the interface **`LogicalTime`** called **`HLAInteger64Time`**. This implementation shall use an **`HLAInteger64BE`** as its underlying representation of time. It shall use 0 as the value of "initialTime" and the largest positive value defined for an **`HLAInteger64BE`** as its final time. Methods that are defined to return or accept a **`LogicalTime`** reference shall actually return or expect a reference to **`HLAInteger64Time`**. Methods defined as returning or accepting a **`LogicalTimeInterval`** reference shall return or expect a reference to **`HLAInteger64TimeInterval`** (see below).

— An implementation of **LogicalTimeFactory** called **HLAInteger64TimeFactory**. All of the methods of this implementation shall return references to **HLAInteger64Time**, as defined above.
— An implementation of **LogicalTimeInterval** called **HLAInteger64TimeInterval**. This implementation shall use an **HLAInteger64BE** as its underlying representation of a time interval. Methods defined as returning or accepting a **LogicalTimeInterval** reference shall return or expect a reference to **HLAInteger64TimeInterval**. The value of epsilon for the **HLAInteger64TimeInterval** shall be represented by 1. When one of the time types is asked to be incremented by epsilon, the implementation shall increment the value by the most appropriate value to create a value that can be differentiated.
— An implementation of **LogicalTimeIntervalFactory** called **HLAInteger64Time-IntervalFactory**. All of the methods of this implementation shall return references to HLA**Integer64TimeInterval** as defined above.

12.4.2 HLAFloat64Time

The implementation of **HLAFloat64Time** shall provide the following:

— An implementation of the interface **LogicalTime** called **HLAFloat64Time**. This implementation shall use an **HLAFloat64BE** as its underlying representation of time. It shall use 0.0 as the value of "initialTime" and the largest finite positive value defined for an **HLAFloat64BE** as its final time. Methods that are defined to return or accept a **LogicalTime** reference shall actually return or expect a reference to **HLAFloat64Time**. Methods defined as returning or accepting a **LogicalTimeInterval** reference shall return or expect a reference to **HLAFloat64TimeInterval** (see below).
— An implementation of **LogicalTimeFactory** called **HLAFloat64TimeFactory**. All of the methods of this implementation shall return references to **HLAFloat64Time**, as defined above.
— An implementation of **LogicalTimeInterval** called **HLAFloat64TimeInterval**. This implementation shall use a **HLAFloat64BE** as its underlying representation of a time interval. Methods defined as returning or accepting a **LogicalTimeInterval** reference shall return or expect a reference to **HLAFloat64TimeInterval**. The value of epsilon for the **HLAFloat64TimeInterval** is a constant; however, it may result in a larger change when used in an addition or subtraction operation. Since a floating point arithmetic operation may require rounding, the result of adding or subtracting epsilon to a **HLAFloat64Time** value may round back to the original value. To prevent this conflict with the statements given in 12.3, the implementation of epsilon in an arithmetic operation shall act as the smallest value that guarantees a change in the result. A side effect of this adjustment to epsilon during an arithmetic operation is that the operation may not be reversible (e.g., given $T' = T + Ie \Rightarrow T' - T >= Ie$).
— An implementation of **LogicalTimeIntervalFactory** called **HLAInteger64Time-Factory**. All of the methods of this implementation shall return references to **HLAInteger64TimeInterval** as defined above.

12.5 Connect

For all of the APIs, each **RTIambassador**/**FederateAmbassador** pair may be part of only one connection to the RTI.

12.6 Concurrency and reentrancy

The following requirements apply to an RTI:

— The RTI shall not make concurrent RTI-initiated service invocations to the same **FederateAmbassador** instance.

— The RTI shall support concurrent joined federate-initiated service invocations to the same RTIambassador instance. The service invocations shall be processed, in parallel or sequentially.
— When in EVOKED mode, all RTI-initiated service invocations shall be made using the same thread context that was used in the invocation of the *Evoke Callback/Evoke Multiple Callbacks* service.
— The RTI shall support concurrent joined federate-initiated service invocations to different RTIambassador instances. The calls shall be processed, in parallel or sequentially.
— An RTIambassador instance shall be capable of creating concurrent federation executions.
— The RTI shall support invocation by a joined federate of a subset of joined federate-initiated services from within the context of an RTI-initiated service invocation on the same joined federate. Invocation of joined federate-initiated services not within this subset shall generate the *Call Not Allowed From Within Callback* exception. The allowed subset of joined federate-initiated services is all joined federate-initiated services except *Connect*, *Disconnect*, *Join Federation Execution*, *Resign Federation Execution*, *Evoke Callback*, and *Evoke Multiple Callbacks*.

12.7 Dynamic link compatibility

The APIs defined in this specification were designed to meet the purpose and objectives for dynamic link compatibility of RTIs stated in 1.2 and 1.3 of SISO-STD-004.1-2004 [B5]. The precise details of dynamic link compatibility are language specific and are described in detail in the appropriate language section.

12.8 *Reflect Attribute Values* service methods

The *Reflect Attribute Values* service corresponds to three overloaded versions of **`reflectAttributeValues()`** in the Java and C++ APIs. Which version the RTI calls is dependent on if the **`updateAttributeValues()`** was called with a timestamp and if the message was received TSO. The optional arguments are encapsulated in a **`SupplementalReflectInfo`** object that is added as an argument to the *Reflect Attribute Values* service method. The values of the convey region designator switch and the Convey Producing Federate Switch control the contents of the **`SupplementalReflectInfo`** object.

The first version passes no optional arguments but does pass a **`SupplementalReflectInfo`** object with possibly valid values. This version is called when there is not a timestamp associated with the **`updateAttributeValues()`** call and the message was received RO. The second version passes the timestamp, receive message order type optional arguments, and a **`SupplementalReflectInfo`** object with possibly valid values. This version is called when there is a timestamp associated with the **`updateAttributeValues()`** call and the message was received RO. The third version passes the timestamp, receive message order type, message retraction designator optional arguments, and a **`SupplementalReflectInfo`** object with possibly valid values. This version is called when there is a timestamp associated with the **`updateAttributeValues()`** call and the message was received TSO.

12.9 *Receive Interaction* service methods

The *Receive Interaction* service corresponds to three overloaded versions of **`receiveInteraction()`** in the Java and C++ APIs. Which version the RTI calls is dependent on if the **`sendInteraction()`** was called with a timestamp and if the message was received TSO. The optional arguments are encapsulated in a **`SupplementalReceiveInfo`** object that is added as an argument to the *Receive Interaction* service method. The values of the convey region designator switch and the Convey Producing Federate Switch control the contents of the **`SupplementalReceiveInfo`** object.

The first version passes no optional arguments but does pass a **`SupplementalReceiveInfo`** object with possibly valid values. This version is called when there is not a timestamp associated with the **`sendInteraction()`** call and the message was received RO. The second version passes the timestamp,

receive message order type optional arguments, and a **SupplementalReceiveInfo** object with possibly valid values. This version is called when there is a timestamp associated with the **sendInteraction()** call and the message was received RO. The third version passes the timestamp, receive message order type, message retraction designator optional arguments, and a **SupplementalReceiveInfo** object with possibly valid values. This version is called when there is a timestamp associated with the **sendInteraction()** call and the message was received TSO.

12.10 *Remove Object Instance* service methods

The *Remove Object Instance* service corresponds to three overloaded versions of the **removeObjectInstance()** in the Java and C++ APIs. Which version the RTI calls is dependent on if the **removeObjectInstance()** was called with a timestamp and if the message was received TSO. The optional arguments are in a **SupplementalRemoveInfo** object that is added as an argument to the *Remove Object Instance* service method. The values of the Convey Producing Federate Switch control the contents of the **SupplementalReceiveInfo** object.

The first version returns no optional arguments but does pass a **SupplementalRemoveInfo** object with possibly valid values. This version is called when there is not a timestamp associated with the **removeObjectInstance()** call and the message was received RO. The second version returns the timestamp, receive message order type optional arguments, and a **SupplementalRemoveInfo** object with possibly valid values. This version is called when there is a timestamp associated with the **removeObjectInstance()** call and the message was received RO. The third version returns the timestamp, receive message order type, message retraction designator optional arguments, and a **SupplementalRemoveInfo** object with possibly valid values. This version is called when there is a timestamp associated with the **removeObjectInstance()** call and the message was received TSO.

12.11 Java

The Java API is composed of interfaces and classes. Where an interface is specified, the RTI implementers shall furnish an implementation of the interface. For instance, **RTIambassador** is specified as an interface. Therefore, an RTI implementer shall furnish an implementation of the interface, for instance:

```
package com.myrticompany.amazingrti;
public final class MyRTIimpl implements hla.rti1516e.RTIambassador
{…}
```

Each such interface as specified remains part of the API. Thus, federate code can use the specified interface type. The **FederateAmbassador**, however, is an exception to this rule and shall be implemented by a federate developer.

Classes defined in the Java API, such as **RTIexception** and **ResignAction**, shall be included in each implementation of the RTI.

The entire Java API is included in the package **hla.rti1516e** and subpackages. Because the specified interfaces and classes constitute the API, the entire package **hla.rti1516e** shall be furnished without additions or deletions or change of package name as part of each implementation. Implementers shall place their implementation classes in another package.

12.11.1 Designators

The Java interfaces that represent handles shall implement a **hashCode()** method that returns an **int** that is suitable for use as a hash value for use in storing handles in a hash table. The hash value may not be unique across two separate handle values but will be equal given two separate instances of the same handle

value. The following are the properties of the hash value where H1 and H2 are separate instances of hash values.

— H1 == H2 implies H1.hashCode() == H2.hashCode()
— H1 != H2 does not imply H1.hashCode() != H2.hashCode()

12.11.2 Dynamic link compatibility

Java dynamic link compatibility for this specification is provided through use of the service registry, as described in the javadoc for ServiceRegistry, available at http://java.sun.com/j2se/1.5.0/docs/api/javax/imageio/spi/ServiceRegistry.html.

The RTI shall provide the HLA 1516-2009.1 Java API classes in the package **`hla.rti1516e`** with corresponding subpackages. Each RTI vendor shall provide their RTI implementation classes in an implementation-specific package.

This standard provides implementations for the classes **`RtiFactoryFactory`**, **`NullFederateAmbassador`**, and **`LogicalTimeFactoryFactory`**.

The RTI shall provide implementations for the Encoding Helper classes in the **`hla.rti1516e.encoding`** package. The RTI shall also provide a class that implements the **`RtiFactory`** interface defined in the **`hla.rti1516e`** package. This class shall reside in an implementation-specific package outside of the **`hla.rti1516e`** hierarchy. The **`RtiFactory`** class shall be listed in a service registry entry, i.e., a file named hla.rti1516e.RtiFactory in the META-INF/services directory of the Java Archive (JAR) shall contain a line with the class name of the implementing class, e.g., com.mycompany.MyRtiFactoryImpl.

The RTI shall also provide implementations of the standardized time classes **`HLAinteger64Time`**, **`HLAinteger64Interval`**, **`HLAfloat64Time`**, and **`HLAfloat64Interval`**, along with their factory classes **`HLAinteger64TimeFactory`** and **`HLAfloat64TimeFactory`**. The time factories shall be listed in a service registry entry, i.e., a file named hla.rti1516e.LogicalTimeFactory in the META-INF/services directory of the JAR shall contain a line with the class name of each implementing class, e.g., com.mycompany.HLAinteger64TimeFactoryImpl and com.mycompany.HLAfloat64Time-FactoryImpl.

All RTIs shall support the interpretation of the FDD designator as a URL.

12.11.2.1 Factory mechanism

An RTI implementation shall furnish an implementation of the **`hla.rti1516e.RtiFactory`** interface.

12.11.2.1.1 Registration

An RTI implementation shall register as a service by creating a file in the services subdirectory of the META-INF directory in the JAR file containing the RTI implementation. The file shall be named hla.rti1516e.RtiFactory and contain a line specifying the name of the class that implements the **`RtiFactory`** interface, e.g.,

```
org.someorganization.anrti.RtiFactoryImpl
```

12.11.2.1.2 RtiFactoryFactory

To acquire a reference to an **`RtiFactory`** implementation, the federate uses the **`RtiFactoryFactory`** class provided with the standard.

```
RtiFactory rtiFactory = RtiFactoryFactory.getRtiFactory();
RTIambassador _rtiAmbassador = rtiFactory.createRtiAmbassador();
```

The federate can acquire a named **RtiFactory**,

```
RtiFactory rtiFactory = RtiFactoryFactory.getRtiFactory("SomeRTI");
RTIambassador _rtiAmbassador = rtiFactory.createRtiAmbassador();
```

or get a list of available **RtiFactories**.

```
Set<RtiFactory> rtiFactories = RtiFactoryFactory.getRtiFactories();
```

12.11.2.1.3 RtiFactory

The **RtiFactory** class or the Java API for this specification follows the RtiFactory class description detailed in 6.6.1 of SISO-STD-004.1-2004 [B5].

In addition, this specification also supplies a **getEncoderFactory()** method that returns the EncoderFactory that can be used by the encoding services described in 12.11.3.2.

12.11.3 Additional functionality

Implementations of the Java API shall add the following utility classes to the **hla.rti1516e** package.

12.11.3.1 NullFederateAmbassador class

The **NullFederateAmbassador** class provides empty implementations of all methods in **FederateAmbassador**.

12.11.3.2 encoding package

The **encoding** package provides a set of classes providing helper functions for encoding datatypes according to IEEE Std 1516.2-2010. These classes are in a package named encoding within the **hla.rti1516e** package (e.g., **hla.rti1516e.encoding**)

All encoding helper classes shall support the encode and decode methods for converting language-specific datatypes to and from standard HLA encoded byte-arrays. All noncomplex encoder classes, as listed in Table 19, shall support the getValue and setValue methods for retrieving and updating the value.

All complex encoder classes shall support elements that make up the complex class. It shall be possible to use both complex and noncomplex encoder class instances as elements of a complex class instance.

A method named add shall be supported for adding elements to an instance of such a complex class. See Table 20. Another method named get(index) for retrieving data elements shall also be supported.

The HLAvariantRecord shall support the method setVariant to associate different encoders/decoders with different discriminants and the methods setDiscriminant and getDiscriminant to indicate which variant of the record that shall be used for composing, encoding, and decoding data elements.

The HLAVariableArray encoder shall provide a size method that provides the number of elements.

The RTI shall supply an implementation of the interface **hla.rti1516e.encoding.EncoderFactory**, a factory that shall provide methods for creating encoding helper instances. The **RtiFactory** shall supply a method getEncoderFactory that returns an EncoderFactory instance.

Table 19—Noncomplex Java encoding helpers

HLA data representation	Encoding helper class	Java type
HLAoctet	HLAoctet	byte
HLAoctetPairBE	HLAoctetPairBE	short
HLAoctetPairLE	HLAoctetPairLE	short
HLAinteger16BE	HLAinteger16BE	short
HLAinteger16LE	HLAinteger16LE	short
HLAinteger32BE	HLAinteger32BE	int
HLAinteger32LE	HLAinteger32LE	int
HLAinteger64BE	HLAinteger64BE	long
HLAinteger64LE	HLAinteger64LE	long
HLAfloat32BE	HLAfloat32BE	float
HLAfloat32LE	HLAfloat32LE	float
HLAfloat64BE	HLAfloat64BE	double
HLAfloat64LE	HLAfloat64LE	double
HLAASCIIchar	HLAASCIIchar	byte
HLAunicodeChar	HLAunicodeChar	short
HLAbyte	HLAbyte	byte
HLAhandle	HLAhandle	byte[]
HLAlogicalTime	HLAlogicalTime	byte[]
HLAlogicalTimeInterval	HLAbyte	byte[]
HLAASCIIString	HLAASCIIString	String
HLAunicodeString	HLAunicodeString	String
HLAopaqueData	HLAopaqueData	byte[]

Table 20—Complex Java encoding helpers

HLA data representation	Encoding helper class
HLAfixedRecord	HLAfixedRecord
HLAvariantRecord	HLAvariantRecord
HLAfixedArray	HLAfixedArray
HLAvariableArray	HLAvariableArray

12.11.4 Services

In general, each service in Clause 4 through Clause 10 corresponds to one method in each API. In some cases, because of optional arguments to a service or programming language constraints, a single service may be mapped to multiple methods in an API.

For the Java API, the federate-initiated services invoked on the RTI are methods of the **`RTIambassador`** interface, and RTI-initiated service callbacks on the federate are methods of the **`FederateAmbassador`** interface. The RTI and federate ambassador services in the Java API that do not follow a one-to-one mapping with the services in Clause 4 through Clause 10 of the specification are described in 12.11.4.1 through 12.11.4.34.

12.11.4.1 *Create Federation Execution* service

The set of FOM module designators argument is implemented as an array of URLs where each item in the array is a URL for a file containing one FOM module.

In addition to the supplied arguments listed with the description of this service, the Java API requires the federate to supply a **`String logicalTimeImplementationName`** that contains the name of the logical time implementation. In addition to the specified exceptions listed with the description of this service, the Java API may also throw a **`CouldNotCreateLogicalTimeFactory`**. This exception indicates that the RTI was not able to create an instance of the logical time factory class that was named in the **`String logicalTimeImplementationName`** argument.

12.11.4.2 *Join Federation Execution* service

The set of FOM module designators argument is implemented as an array of URLs where each item in the array is a URL for a file containing one FOM module.

In addition to the specified exceptions listed with the description of this service, the Java API may also throw a **`CouldNotCreateLogicalTimeFactory`**. This exception indicates that the RTI was not able to create an instance of the logical time factory class that was named in the **`String logicalTimeImplementationName`** argument to the *Create Federation Execution* service.

12.11.4.3 *Register Federation Synchronization Point* service

The optional argument representing the set of joined federate designators is implemented in the Java API as two **`registerFederationSynchronizationPoint()`** methods: one that takes a **`Federate-HandleSet`** as an argument, and one that does not.

12.11.4.4 *Confirm Synchronization Point Registration †* service

The registration-success indicator is implemented in the Java API as two distinct methods: **`synchronizationPointRegistrationSucceeded()`** and **`synchronizationPoint-RegistrationFailed()`**. A Synchronization Point Failure Reason (discussed in 12.11.5.10), in the form of a **`SynchronizationPointFailureReason`**, is passed as an argument to **`synchronizationPointRegistrationFailed()`**.

12.11.4.5 *Synchronization Point Achieved* service

The optional *synchronization-success indicator* parameter is implemented as two methods, one of which takes a boolean as an argument.

12.11.4.6 *Request Federation Save* service

The optional timestamp argument is implemented as two methods, one of which takes a **`LogicalTime`** as an argument.

12.11.4.7 *Initiate Federate Save* † service

The optional timestamp argument is implemented as two methods, one of which takes a **`LogicalTime`** as an argument.

12.11.4.8 *Federate Save Complete* service

This service corresponds to two methods in the Java API: **`federateSaveComplete()`** and **`federateSaveNotComplete()`**.

12.11.4.9 *Federation Saved* † service

This service corresponds to two methods in the Java API: **`federationSaved()`** and **`federationNotSaved()`**. A Save Failure Reason (discussed in 12.11.5.11), in the form of a **`SaveFailureReason`**, is passed as an argument to **`federationNotSaved()`**.

12.11.4.10 *Confirm Federation Restoration Request* † service

This service is mapped to the following two methods in the Java API: **`requestFederationRestore-Succeeded()`** and **`requestFederationRestoreFailed()`**.

12.11.4.11 *Federate Restore Complete* service

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

12.11.4.12 *Federation Restored* † service

This service corresponds to two methods in the Java API: **`federationRestored()`** and **`federationNotRestored()`**. A Restore Failure Reason (discussed in 12.11.5.12), in the form of a **`RestoreFailureReason`**, is passed as an argument to **`federationNotRestored()`**.

12.11.4.13 *Connect* service

In addition to the supplied arguments listed with the description of this service, the Java API requires the federate to supply an instance of the **`interface FederateAmbassador`**. The optional **`localSettingsDesignator`** parameter is required and can contain RTI implementation-specific configuration information. An empty string for the **`localSettingsDesignator`** parameter shall be used to indicate the default local settings.

12.11.4.14 *Unpublish Object Class Attributes* service

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

12.11.4.15 *Subscribe Object Class Attributes* service

The optional passive subscription indicator and optional update rate designator are implemented in the Java API as two overloaded methods: **`subscribeObjectClassAttributes()`** and **`subscribeObjectClassAttributesPassively()`**. One version of each method has only the required arguments, and the second version adds the optional **`String updateRateDesignator`** argument.

12.11.4.16 *Unsubscribe Object Class Attributes* service

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

12.11.4.17 *Subscribe Interaction Class* service

The optional passive subscription indicator is implemented in the Java API as two methods: **`subscribeInteractionClass()`** and **`subscribeInteractionClassPassively()`**.

12.11.4.18 *Reserve Object Instance Name* service

The optional single name parameter or list of names parameter is implemented in the Java API as two methods: **`reserveObjectInstanceName()`** and **`reserveMultipleObjectInstanceName()`**.

12.11.4.19 *Object Instance Name Reserved †* service

This service corresponds to two methods in the Java API: **`objectInstanceNameReservationSucceeded()`** and **`objectInstanceNameReservationFailed()`**.

12.11.4.20 *Multiple Object Instance Name Reserved †* service

This service corresponds to two methods in the Java API: **`multipleObjectInstanceNameReservationSucceeded()`** and **`multipleObjectInstanceNameReservationFailed()`**.

12.11.4.21 *Register Object Instance* service

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

12.11.4.22 *Discover Object Instance* service

The optional argument representing the producing joined federate designator is implemented in the Java API as two **`discoverObjectInstance()`** methods: one that takes a **`FederateHandle`** as an argument for the producing joined federate designator, and one that does not.

12.11.4.23 *Update Attribute Values* service

The optional argument representing the timestamp is implemented in the Java API as two **`updateAttributeValues()`** methods: one that takes a **`LogicalTime`** as an argument for the timestamp, and one that does not. The version that takes a **`LogicalTime`** also returns a **`MessageRetractionReturn`** for the optional message retraction designator.

12.11.4.24 *Reflect Attribute Values †* service

This service corresponds to three overloaded versions of the **`reflectAttributeValues()`** method in the Java API. Which version the RTI calls is dependent on if the **`updateAttributeValues()`** was called with a timestamp and if the message was received TSO. See 12.8 for details how to determine which version the RTI will call.

12.11.4.25 *Send Interaction* service

The optional argument representing the timestamp is implemented in the Java API as two **`sendInteraction()`** methods: one that takes a **`LogicalTime`** as an argument for the timestamp, and one that does not. The version that takes a **`LogicalTime`** also returns a **`MessageRetractionReturn`** for the optional message retraction designator.

12.11.4.26 *Receive Interaction †* service

This service corresponds to three overloaded versions of the **`receiveInteraction()`** method in the Java API. Which version the RTI calls is dependent on if the **`sendInteraction()`** was called with a timestamp and if the message was received TSO. See 12.9 for details on how to determine which version the RTI will call.

12.11.4.27 *Delete Object Instance* service

The optional argument representing the timestamp is implemented in the Java API as two **`deleteObjectInstance()`** methods: one that takes a **`LogicalTime`** as an argument for the timestamp, and one that does not. The version that takes a **`LogicalTime`** also returns a **`MessageRetractionReturn`** for the optional message retraction designator.

12.11.4.28 *Remove Object Instance †* service

This service corresponds to three overloaded versions of the **`removeObjectInstance()`** method in the Java API. See 12.10 for details on how to determine which version the RTI will call.

12.11.4.29 *Request Attribute Value Update* service

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

12.11.4.30 *Inform Attribute Ownership †* service

In the Java API, 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.

12.11.4.31 *Register Object Instance With Regions* service

This service corresponds to two overloaded versions of the **`registerObjectInstance-WithRegions()`** method in the Java API. The second version takes a **`String`** as an argument to represent the optional object instance name.

12.11.4.32 *Subscribe Object Class Attributes With Regions* service

The optional passive subscription indicator and optional update rate designator are implemented in the Java API as two overloaded methods: **`subscribeObjectClassAttributesWithRegions()`** and **`subscribeObjectClassAttributesPassivelyWithRegions()`**. One version of each method has only the required arguments, and the second version adds the optional **`String updateRateDesignator`** argument.

12.11.4.33 *Subscribe Interaction Class With Regions* service

The optional passive subscription indicator is implemented in the Java API as two methods: **`subscribeInteractionClassWithRegions()`** and **`subscribeInteractionClass-PassivelyWithRegions()`**.

12.11.4.34 *Send Interaction With Regions* service

The optional argument representing the timestamp is implemented in the Java API as two **`sendInteractionWithRegions()`** methods: one that takes a **`LogicalTime`** as an argument for the timestamp, and one that does not. The version that takes a **`LogicalTime`** also returns a **`MessageRetractionReturn`** for the optional message retraction designator.

12.11.5 Abstract datatypes

In 1.4.2, general conventions used to describe the abstract datatypes used in Clause 4 through Clause 10 are presented. In 12.11.5.1 through 12.11.5.26, the abstract datatypes are further detailed by mapping an individual datatype to its implementation in the Java API. Note that the datatypes from Clause 11 are explained in that clause and are not repeated here. Additionally, designators are a subject in their own right and are covered in 12.2.

Abstract datatypes are mapped to idiomatic Java types (if possible, to a type in the Java Platform APIs). Java 5 is assumed. This is especially evident in the use of the Java Development Kit Collections framework. All interfaces extend, and classes implement, **`java.io.Serializable`**. The abstract datatypes make use of generics (typed collections) to achieve type safety in collections.

12.11.5.1 Names, labels, and strings

The names, labels, and strings described in Clause 4 through Clause 10 of this specification are mapped to **`java.lang.String`**.

12.11.5.2 Boolean values

The Java primitive type **`boolean`** is used to represent all boolean values.

12.11.5.3 Exceptions

All exceptions in the Java API inherit from a class named **`RTIexception`**, which extends **`java.lang.Exception`**. This hierarchy allows all exceptions in the Java API to be caught as a single base class.

12.11.5.4 Handles

Most designators in the Java API are handles. Each handle is a distinct Java interface that shall be implemented by the RTI implementer. Each handle interface supports **`equals()`** and **`hashCode()`** so that handles can be used as keys in **`HashTables`**. Handles of different types are incommensurable.

Handle instances typically are returned from RTI services, e.g., **`getAttributeHandle()`**. These services return a reference to the handle interface and, thus, hide the implementation class. The federate programmer typically uses only handle references that have been returned by the RTI.

Each handle interface supports **`encode()`**, which shall place a compact, opaque representation suitable for network transmission into the provided **`byte`** array at the specified offset. Each handle interface has a corresponding handle factory interface, which shall also be implemented by the RTI implementer. The factory interface defines **`decode()`**, which returns a new handle instance from a representation in the provided **`byte`** array at the indicated offset. 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.

Handle classes also provide a **`toString()`** 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, employ the appropriate support service from Clause 10.

12.11.5.5 Resignation directive

The *Resign Federation Execution* service allows the specification of one of six directives, which are represented by an enumeration, **`ResignAction`**. The six different directives of the *Resign Federation Execution* service are defined as the six enumerated values that instances of the **`ResignAction`** enumeration may take. The **`ResignAction`** enumeration has the following values:

— **UNCONDITIONALLY_DIVEST_ATTRIBUTES**
— **DELETE_OBJECTS**
— **CANCEL_PENDING_OWNERSHIP_ACQUISITIONS**
— **DELETE_OBJECTS_THEN_DIVEST**
— **CANCEL_THEN_DELETE_THEN_DIVEST**
— **NO_ACTION**

12.11.5.6 Attribute and parameter values and user-supplied tag

Several services in the HLA interface allow the federate programmer to supply additional information to be transported by the RTI in an opaque manner, such as the *Update Attribute Values* service. Attribute and parameter values are represented as **`byte`** arrays, **`byte[]`**. **`byte[]`** is also used to supply or deliver user-supplied tags.

12.11.5.7 Sets of designators

Sets of designators, e.g., federate designators in the *Register Federation Synchronization Point* service or attribute designator sets in the *Publish Object Class Attributes* service, are represented by interfaces that extend **`java.util.Set`** for the respective handle type. This interface is part of the Java 2 Collections framework. The RTI implementer's implementation of each such set shall adhere to the following:

— Members shall be of the appropriate type, e.g., **`AttributeHandle`** for **`AttributeHandleSet`**. Attempts to add an instance of an inappropriate type shall elicit **`IllegalArgumentException`**.
— Implementations of the sets shall implement all defined operations (none are optional).
— Methods that might take an incommensurable set as an actual parameter shall throw **`IllegalArgumentException`**.

Each such handle set has a corresponding factory interface. Thus, the implementation class shall be hidden from the federate programmer. A set of handles is provided for federate handles, attribute handles, dimension handles, and region handles.

12.11.5.8 Success indicators

Several of the requests a federate may make of an RTI shall be acknowledged by the RTI to the federate. This acknowledgment can indicate that the request succeeded or failed. In the Java API, each request acknowledgment corresponds to a positive and a negative callback method on the **`FederateAmbassador`**, e.g., **`federationSaved()`** and **`federationNotSaved()`**, **`requestFederationRestoreSucceeded()`** and **`requestFederationRestoreFailed()`**, and **`federationRestored()`** and **`federationNotRestored()`**.

12.11.5.9 Callback Model

When the **`connect()`** method is invoked by a federate, it shall supply an instance of the **`CallbackModel`** enumeration as an argument. The **`CallbackModel`** enumeration provides the following values:

— **HLA_IMMEDIATE**
— **HLA_EVOKED**

12.11.5.10 Synchronization Point Failure Reason

When the **`synchronizationPointRegistrationFailed()`** method is invoked by the RTI on a federate, the RTI provides an instance the **`SynchronizationPointFailureReason`** enumeration as an argument. The **`SynchronizationPointFailureReason`** enumeration has the following values:

— **SYNCHRONIZATION_POINT_LABEL_NOT_UNIQUE**
— **SYNCHRONIZATION_SET_MEMBER_NOT_JOINED**

12.11.5.11 Save Failure Reason

When the **`federationNotSaved()`** method is invoked by the RTI on a federate, the RTI provides an instance of the **`SaveFailureReason`** enumeration as an argument. The **`SaveFailureReason`** enumeration has the following values:

— **RTI_UNABLE_TO_SAVE**
— **FEDERATE_REPORTED_FAILURE_DURING_SAVE**
— **FEDERATE_RESIGNED_DURING_SAVE**
— **RTI_DETECTED_FAILURE_DURING_SAVE**
— **SAVE_TIME_CANNOT_BE_HONORED**
— **SAVE_ABORTED**

12.11.5.12 Restore Failure Reason

When the **`federationNotRestored()`** method is invoked by the RTI on a federate, the RTI provides an instance of the **`RestoreFailureReason`** enumeration as an argument. The **`RestoreFailureReason`** enumeration has the following values:

— **RTI_UNABLE_TO_RESTORE**
— **FEDERATE_REPORTED_FAILURE_DURING_RESTORE**
— **FEDERATE_RESIGNED_DURING_RESTORE**
— **RTI_DETECTED_FAILURE_DURING_RESTORE**

— **RESTORE_ABORTED**

12.11.5.13 List of joined federates save status

The list of joined federates and the save status of each is implemented as an array of **`FederateHandleSaveStatusPair`** objects. Each **`FederateHandleSaveStatusPair`** object contains a **`FederateHandle`** and a **`SaveStatus`** representing the save status for that joined federate. The **`SaveStatus`** enumeration has the following values:

— **NO_SAVE_IN_PROGRESS**
— **FEDERATE_INSTRUCTED_TO_SAVE**
— **FEDERATE_SAVING**
— **FEDERATE_WAITING_FOR_FEDERATION_TO_SAVE**

12.11.5.14 List of joined federates restore status

The list of joined federates and the restore status of each is implemented as an array of **`FederateRestoreStatus`** objects. Each **`FederateRestoreStatus`** object contains a **`FederateHandle`** that contains the pre-restore federate handle, a **`FederateHandle`** that contains the post-restore federate handle, and a **`RestoreStatus`** representing the restore status for that joined federate. The **`RestoreStatus`** enumeration has the following values:

— **NO_RESTORE_IN_PROGRESS**
— **FEDERATE_RESTORE_REQUEST_PENDING**
— **FEDERATE_WAITING_FOR_RESTORE_TO_BEGIN**
— **FEDERATE_PREPARED_TO_RESTORE**
— **FEDERATE_RESTORING**
— **FEDERATE_WAITING_FOR_FEDERATION_TO_RESTORE**

12.11.5.15 Passive subscription indicator

The *Subscribe Object Class Attributes, Subscribe Interaction Class*, *Subscribe Object Class Attributes With Regions*, and *Subscribe Interaction Class With Regions* services each may optionally provide a boolean value to indicate whether the subscription is active. In the Java API, this corresponds to a pair of methods on **`RTIambassador`**, e.g., **`subscribeObjectClassAttributes()`** and **`subscribeObjectClassAttributesPassively()`**.

12.11.5.16 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 **`AttributeHandleValueMap`**. This interface extends **`java.util.Map`**, part of the Java 2 Collections framework. The RTI implementer's implementation of **`AttributeHandleValueMap`** shall adhere to the following:

— The keys are instances of **`AttributeHandle`**; **`IllegalArgumentException`** shall be thrown for violations.
— The values are instances of **`byte[]`**; **`IllegalArgumentException`** shall be thrown for violations.
— All **`java.util.Map`** operations shall be implemented; none are optional.
— The implementation shall not accept null mappings, i.e., a mapping to a **`null`** value.

The interface **`AttributeHandleValueMapFactory`** shall be implemented, and the implementation class for **`AttributeHandleValueMap`** shall be hidden from the federate programmer. The federate

programmer shall obtain a reference to a factory instance by the method **`getAttributeHandleValueMapFactory()`** on **`RTIambassador`**.

12.11.5.17 Constrained set of interaction parameter designator and value pairs

The *Send Interaction, Receive Interaction †*, and *Send Interaction With Regions* services require as arguments a constrained set of parameter designator and value pairs. Such sets are instances of **`ParameterHandleValueMap`**. This interface extends **`java.util.Map`**, part of the Java 2 Collections framework. The RTI implementer's implementation of **`ParameterHandleValueMap`** shall adhere to the following:

- The keys are instances of **`ParameterHandle`**; **`IllegalArgumentException`** shall be thrown for violations.
- The values are instances of **`byte[]`**; **`IllegalArgumentException`** shall be thrown for violations.
- All **`java.util.Map`** operations shall be implemented; none are optional.
- The implementation shall not accept null mappings, i.e., a mapping to a **`null`** value.

The interface **`ParameterHandleValueMapFactory`** shall be implemented, and the implementation class for **`ParameterHandleValueMap`** shall be hidden from the federate programmer. The RTI shall provide the federate programmer a reference to a factory instance by the method **`getParameterHandleValueMapFactory()`** on **`RTIambassador`**.

12.11.5.18 Message order type

Message order types are represented by an enumerated type. The following enumerations are provided: **`RECEIVE`** and **`TIMESTAMP`**. These enumerations can be passed to the RTI or compared to an instance of an **`OrderType`** object returned by the RTI.

The **`OrderType`** enumeration shares some aspects of the interface to handles (see 12.11.5.4); specifically, the means to encode an order type, reconstitute an encoded order type, and produce a printable form of an order type are provided. Encoding order types and reconstitution of encoded order types work in the same way as for handles, with the exception that the **`decode()`** method is provided as a **`static`** member of the **`OrderType`** enumerated type rather than as a member of a corresponding factory. Producing a printable form works the same as for handles. The **`toString()`** method is used, and the string produced is not guaranteed to be the same as the name to which the order type corresponds (i.e., the name of the order type as found in the FDD). If the user needs the name form of an order type, employ the *Get Order Name* service.

12.11.5.19 Transportation type

The Java API uses a handle, **`TransportationTypeHandle`**, to represent all of the arguments of type **`TransportationType`** in the API methods. To acquire a handle, use the *Get Transportation Type Handle* service. All RTIs shall support the transportation types **`HLA_RELIABLE`** and **`HLA_BEST_EFFORT`**.

It is possible that some implementations of the RTI will allow additional transportation types. In this case, those RTI implementations shall accept the new transportation type name in the FDD and the implementation shall

- Return a valid *Transportation Type Handle* for the *Get Transportation Type Handle* service.
- Return a valid *Transportation Type Name* for the *Get Transportation Type Name* service.

12.11.5.20 Ownership designator

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

12.11.5.21 Definition indicator

The *Query GALT* and *Query LITS* services are required to return an indicator to define whether their return value is valid. Both services return a value of type **`TimeQueryReturn`**. This structure contains a **`boolean`** member that indicates whether the **`LogicalTime`** is valid. If the member **`timeIsValid`** is **`false`**, the member **`time`** should not be used. **`TimeQueryReturn`** also supports **`equals()`** and **`hashCode()`** so that **`TimeQueryReturn`** objects can be used as keys in **`HashTable`**s.

12.11.5.22 Optional message retraction designator

The *Update Attribute Values*, *Send Interaction*, *Send Interaction With Regions*, and *Delete Object Instance* services may return a message retraction designator under certain circumstances. In the Java API, this indicator is implemented much like the definition indicator described above. The methods corresponding to these services always return a **`MessageRetractionReturn`**. The member **`handle`** of this structure should be used only if the member **`retractionHandleIsValid`** is **`true`**.

12.11.5.23 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. In the Java API, this set corresponds to the interface **`AttributeSetRegionSetPairList`**. This interface extends **`java.util.List`**, part of the Java 2 Collections framework. The RTI implementer's implementation of AttributeSetRegionSetPairList shall adhere to the following:

- The elements are instances of the class **`AttributeRegionAssociation`**; **`IllegalArgumentException`** shall be thrown for violations.
- All **`java.util.List`** operations shall be implemented; none are optional.
- The implementation shall not accept null elements; i.e., an element whose value is **`null`**.
- No operation shall accept an incommensurate type; **`IllegalArgumentException`** shall be thrown for violations.

12.11.5.24 Logical time, timestamps, and lookahead

In the Java API, the abstract interfaces for the logical time classes are defined as Java interfaces. The **`LogicalTime`** and **`LogicalTimeInterval`** interfaces extend the **`Comparable`** interface. **`LogicalTime`** and **`LogicalTimeInterval`** interfaces shall have a **`toString()`** method, which yields the representation suitable for printing. The variable array used with the encoding and decoding methods shall be a **`byte`** array.

In addition to the methods mentioned in 12.3, the Java **`LogicalTime`** interface provides the following methods:

- **`hashCode()`** that returns an integer value that shall be equal for any two **`LogicalTime`** instances for which **`equals()`** returns true.
- **`encodedLength()`** that shall return the size of the buffer required to encode the object

— **encode()** that takes a **byte** array and an offset and shall encode the object into the **byte** array at the offset position

In addition to the methods mentioned in 12.3, the Java **LogicalTimeInterval** interface provides the following methods:

— **hashCode()** that returns an integer value that shall be equal for any two **LogicalTime** instances for which **equals()** returns true.
— **encodedLength()** that shall return the size of the buffer required to encode the object
— **encode()** that takes a **byte** array and an offset and shall encode the object into the **byte** array at the offset position

In addition to the methods mentioned in 12.3, the Java **LogicalTimeFactory** interface provides the following methods:

— **decodeTime()** that takes a **byte** array with an offset and returns a **LogicalTime** instance with the value that is decoded from that offset in the **byte** array. This method can throw a **CouldNotDecode** exception.
— **decodeInterval()** that takes a **byte** array with an offset and returns a **LogicalTimeInterval** instance with the value that is decoded from that offset in the **byte** array. This method can throw a **CouldNotDecode** exception.
— **getName()** that returns the **String** name of the logical time implementation that was passed to the *Create Federation Execution* service.

12.11.5.24.1 Registration

A time implementation shall register as a service by creating a file in the services subdirectory of the META-INF directory in the JAR file containing the time implementation. The file shall be named **hla.rti1516e.LogicalTimeFactory** and contain a line specifying the name of the class that implements the **LogicalTimeFactory** interface, e.g.,

```
org.someorganization.time.HLAinteger64TimeFactoryImpl
```

12.11.5.24.2 Standardized time types

An RTI implementation shall provide implementations of the standardized time types as classes implementing the interfaces **HLAinteger64Time**, **HLAinteger64Interval**, **HLAinteger64-TimeFactory**, **HLAfloat64Time**, **HLAfloat64Interval**, and **HLAfloat64TimeFactory**. The **getName()** methods of the **HLAinteger64TimeFactory** and **HLAfloat64TimeFactory** implementations shall return the symbolic names “HLAinteger64Time” and “HLAfloat64Time,” respectively. The **HLAinteger64TimeFactory** and **HLAfloat64TimeFactory** implementations shall be registered according to 12.11.5.24.1.

12.11.5.24.3 LogicalTimeFactoryFactory

To acquire a **LogicalTimeFactory** implementation, the federate may use the **LogicalTimeFactoryFactory** class, provided with this standard, and supply the symbolic name,

```
HLAinteger64TimeFactory logicalTimeFactory = (HLAinteger64TimeFactory)-
LogicalTimeFactoryFactory.getLogicalTimeFactory("HLAinteger64Time");
```

or supply the interface that the time factory shall implement.


```
HLAinteger64TimeFactory logicalTimeFactory = LogicalTimeFactory-
Factory.getLogicalTimeFactory(HLAinteger64TimeFactory.class);
```

The federate may also refer directly to the implementation provided by the RTI,

```
HLAinteger64TimeFactory logicalTimeFactory = new org.someorganization.-
time.HLAinteger64TimeFactoryImpl();
```

or get a list of available LogicalTimeFactories.

```
Set<LogicalTimeFactory> logicalTimeFactories = LogicalTimeFactory-
Factory.getLogicalTimeFactories();
```

12.11.5.24.4 Custom time types

An RTI implementation or a federation implementer may provide implementations of custom time types. These implementations shall be registered according to 12.11.5.24.1.

12.11.5.25 Dimension upper bound and range lower and upper bounds

Several DDM-related services allow bounds to be gotten 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 the Java API using a **`long`** integer. The *Get Dimension Upper Bound* service, as it is getting a single bound value, returns a **`long`**. The *Get Range Bounds* and *Set Range Bounds* services, as they are getting and setting a pair of values, actually return and take a **`RangeBounds`** object, which is composed of a pair of **`long`**s, **`lower`** and **`upper`**.

12.11.5.26 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 in the Java API using a **`double`**.

12.11.6 Memory ownership semantics

The following are memory management conventions for parameters to methods on **`RTIambassador`** and **`FederateAmbassador`**.

All Java parameters, including object references, are passed by value. Therefore, there is no need to specify further conventions for primitive types.

Unless otherwise noted, reference parameters adhere to the following convention:

— The referenced object is created (or acquired) by the caller. The callee must copy during the call anything it wishes to save beyond the completion of the call.
— Unless otherwise noted, a reference returned from a method represents a new object created by the callee. The caller is free to modify the object whose reference is returned.

12.12 C++

The C++ API is intended to achieve several goals as follows:

— To provide as natural a mapping to the abstract interface specification as possible.

— To prevent as many common federate programming errors as practical. This goal includes compile-time type safety as well as dynamic memory management errors.
— To provide flexibility to RTI implementers while adhering to a common API (as used by the federate programmer) and without adding significant overhead.
— To support dynamic link compatibility that allows federates to switch between RTI implementations without recompiling.

The following subclauses elaborate on some of these details, from the perspective of the federate programmer. This discussion is not intended as a guide for RTI implementation.

12.12.1 Dynamic link compatibility

C++ dynamic link compatibility for this specification follows the technical approach detailed in Clause 5 of SISO-STD-004.1-2004 [B5].

The implementation of the RTI API shall be provided in a library named librti1516e followed by the extension appropriate for each platform, e.g., librti1516e.dll or librti1516e.so. A 64-bit version of the library shall have a '64' suffix, e.g., librti1516e64.so. The implementation may provide a debug version of the library with a 'd' suffix, e.g., librti1516ed.dll or librti1516e64d.dll. The librti1516e library shall contain all classes and method implementations except **`rti1516e::LogicalTimeFactoryFactory`**. The librti1516e library shall contain implementations of the standardized time classes **`HLAinteger64Time`**, **`HLAinteger64Interval`**, **`HLAfloat64Time`**, and **`HLAfloat64Interval`**, along with their factory classes **`HLAinteger64TimeFactory`** and **`HLAfloat64TimeFactory`**. The librti1516e library shall also contain an implementation of the class **`rti1516e::HLAlogicalTimeFactory-Factory`** that provides a factory for the standardized time types.

The implementation of **`rti1516e::LogicalTimeFactoryFactory`** shall be contained in a federate-supplied library named libfedtime1516e, followed by the extension appropriate for each platform, 64-bit version, and debug version, e.g., libfedtime1516e.dll or libfedtime1516e64d.so. An RTI uses the static member function **`rti1516e::LogicalTimeFactoryFactory::makeLogicalTime-Factory()`** to create an instance of an **`rti1516e::LogicalTimeFactory`**. That factory is used in turn to create instances of federation-provided subclasses of **`rti1516e::LogicalTime`** and **`rti1516e::LogicalTimeInterval`**. The **`rti1516e::LogicalTimeFactoryFactory`** shall forward requests for other time types, e.g., the standardized time types, to **`rti1516e::HLAlogicalTimeFactoryFactory`**.

An RTI shall provide implementations for all non-pure-virtual functions and classes listed in the RTI header files included in the appendix to this document, with the exception of **`rti1516e::LogicalTime-FactoryFactory`**.

RTI implementations shall link directly with any required third-party libraries. This action ensures that federate developers need to link only to librti1516e and libfedtime1516e in order to use the RTI and ensures that a federate will be able to switch from one RTI implementation to another without having to re-link against the third-party libraries required by the new implementation.

All RTIs shall support the interpretation of the FDD designator as a regular file name.

There are several functions within the RTI API that require the use of the ostream class that is part of the C++ standard library. The legacy version of ostream, which is in the global namespace, is not supported by this API.

12.12.1.1 Factory mechanism

Implementations of the C++ API shall contain a definition of a **`RTIambassadorFactory`** class in the rti1516e namespace that contains a public **`createRTIambassador`** method. This method shall take a **`std::vector`** of **`std::wstring`** as an argument and return an **`std::auto_ptr<RTIambassador>`**. The possible values for the strings in the vector argument are implementation dependent.

12.12.2 Memory ownership semantics

Many programming errors in C++ can be traced back to the improper use of the **`delete`** operator. Frequently, this tendency is a result of poor programming practice, rather than an inherent limitation of the C++ programming language. Considerable effort has been made to prevent memory corruption and leaks resulting from the misuse of the **`delete`** operator in the C++ API.

In particular, the federate programmer never explicitly deletes objects exchanged with the RTI. Simple objects are passed by value. More complex objects are either passed by constant reference (i.e., **`T const &`**) or passed as an **`auto_ptr<T>`** object. In all three cases, the C++ compiler prevents the federate programmer from calling the delete operator on the object.

As explained in 12.12.6, the **`auto_ptr<T>`** implements the strong memory ownership and transfer semantics of the **`std::auto_ptr<T>`** class[9].

12.12.3 Designators

The C++ interfaces that represent handles shall implement a **`hash()`** method that returns a **`long`** that is suitable for use as a hash value for use in storing handles in a hash table. The hash value may not be unique across two separate handle values but will be equal given two separate instances of the same handle value. The following are the properties of the hash value where H1 and H2 are separate instances of hash values.

— H1 == H2 implies H1.hashCode() == H2.hashCode()
— H1 != H2 does not imply H1.hashCode() != H2.hashCode()

12.12.4 Additional functionality

Implementations of the C++ API shall add the utility classes described in 12.12.4.1 and 12.12.4.2 to the rti1516e namespace.

12.12.4.1 `NullFederateAmbassador` class

The **`NullFederateAmbassador`** class provides empty implementations of all methods in **`FederateAmbassador`**.

12.12.4.2 Encoding helper classes

The encoding helper classes provide a set of classes providing helper functions for encoding datatypes according to IEEE Std 1516.2-2010.

All encoding helper classes shall support the encode and decode methods for converting language-specific datatypes to and from standard HLA encoded byte-arrays. All noncomplex encoder classes, as listed Table 21, shall support conversion operator and operator= methods for retrieving and updating the value.

[9] See Stroustrup [B6] for a detailed description.

Table 21—Noncomplex C++ encoding helpers

HLA data representation	Encoding helper class	C++ type/macro
HLAoctet	HLAoctet	Octet
HLAoctetPairBE	HLAoctetPairBE	OctetPair
HLAoctetPairLE	HLAoctetPairLE	OctetPair
HLAinteger16BE	HLAinteger16BE	Integer16
HLAinteger16LE	HLAinteger16LE	Integer16
HLAinteger32BE	HLAinteger32BE	Integer32
HLAinteger32LE	HLAinteger32LE	Integer32
HLAinteger64BE	HLAinteger64BE	Integer64
HLAinteger64LE	HLAinteger64LE	Integer64
HLAfloat32BE	HLAfloat32BE	float
HLAfloat32LE	HLAfloat32LE	float
HLAfloat64BE	HLAfloat64BE	double
HLAfloat64LE	HLAfloat64LE	double
HLAASCIIchar	HLAASCIIchar	char
HLAunicodeChar	HLAunicodeChar	wchar_t
HLAbyte	HLAbyte	char
HLAhandle	HLAhandle	char
HLAlogicalTime	HLAlogicalTime	char
HLAlogicalTimeInterval	HLAlogicalTimeInterval	char
HLAASCIIString	HLAASCIIString	std::string
HLAunicodeString	HLAunicodeString	std::wstring
HLAopaqueData	HLAopaqueData	HLAopaqueData

All complex encoder classes shall support elements that make up the complex class. It shall be possible to use both complex and noncomplex encoder class instances as elements of a complex class instance.

A method named addElement shall be supported for adding elements to an instance of such a complex class. See Table 22. Another method named get(index) for retrieving data elements shall also be supported.

The HLAvariantRecord shall support the method addMapping to associate different encoders/decoders with different discriminants and the methods setDiscriminant and getDiscriminant to indicate which variant of the record that shall be used for composing, encoding, and decoding data elements.

The HLAVariableArray encoder shall provide a size method that provides the number of elements.

Table 22—Complex C++ encoding helpers

HLA data representation	Encoding helper class
HLAfixedRecord	HLAfixedRecord
HLAvariantRecord	HLAvariantRecord
HLAfixedArray	HLAfixedArray
HLAvariableArray	HLAvariableArray

12.12.5 Services

In general, each service in Clause 4 through Clause 10 corresponds to one method in each API. In some cases, because of optional arguments to a service or programming language constraints, a single service may be mapped to multiple methods in an API.

For the C++ API, the federate-initiated services invoked upon the RTI are members of the **`RTIambassador`** class, and RTI-initiated service callbacks on the federate are members of the **`FederateAmbassador`** class. The RTI and federate ambassador services in the C++ API that do not follow a one-to-one mapping with the services in Clause 4 through Clause 10 of the specification are described in 12.12.5.1 through 12.12.5.34.

12.12.5.1 *Create Federation Execution* section

The set of FOM module designators argument is implemented as a **`std::vector`** of **`std::wstrings`** where each item in the vector is a full path to a file containing one FOM module. The optional MIM designator is implemented as an argument with a default value of the empty string.

In addition to the supplied arguments listed with the description of this service, the C++ API requires the federate to supply a **`wstring logicalTimeImplementationName`** that contains the name of the logical time implementation. In addition to the specified exceptions listed with the description of this service, the C++ API may also throw a **`CouldNotCreateLogicalTimeFactory`** exception. This exception indicates that the RTI was not able to create an instance of the logical time factory class that was named in the **`wstring logicalTimeImplementationName`** argument.

12.12.5.2 *Join Federation Execution* section

The optional set of FOM module designators argument is implemented as a **`std::vector`** of **`std::wstrings`** where each item in the vector is a full path to a file containing one FOM module. If the federate does not need to load additional FOM modules, then an empty **`std::vector`** should be passed to the **`joinFederationExecution`** method.

In addition to the specified exceptions listed with the description of this service, the C++ API may also throw a **`CouldNotCreateLogicalTimeFactory`** or **`CallNotAllowedFromWithinCallback`** exception. The **`CouldNotCreateLogicalTimeFactory`** exception indicates that the RTI was not able to create an instance of the logical time factory class that was named in the **`wstring logicalTimeImplementationName`** argument to the *Create Federation Execution* service. The **`CallNotAllowedFromWithinCallback`** exception indicates that this RTI ambassador service call cannot be made while the federate is processing a federate ambassador callback.

12.12.5.3 *Register Federation Synchronization Point* service

The optional argument representing the set of joined federate designators is implemented in the C++ API as two **`registerFederationSynchronizationPoint()`** methods: one that takes an **`FederateHandleSet`** as an argument, and one that does not.

12.12.5.4 *Confirm Synchronization Point Registration †* service

The registration-success indicator is implemented in the C++ API as two distinct methods: **`synchronizationPointRegistrationSucceeded()`** and **`synchronizationPoint-RegistrationFailed()`**. A synchronization failure reason (discussed in 12.12.6.10) is passed as an argument to **`synchronizationPointRegistrationFailed()`**.

12.12.5.5 *Synchronization Point Achieved* service

The optional *synchronization-success indicator* parameter is a required argument in the C++ API.

12.12.5.6 *Request Federation Save* service

The optional timestamp argument is implemented as two methods, one of which takes an **`LogicalTime`** as an argument.

12.12.5.7 *Initiate Federate Save †* service

The optional timestamp argument is implemented as two methods, one of which takes an **`LogicalTime`** as an argument.

12.12.5.8 *Federate Save Complete* service

This service corresponds to two methods in the C++ API: **`federateSaveComplete()`** and **`federateSaveNotComplete()`**.

12.12.5.9 *Federation Saved †* service

This service corresponds to two methods in the C++ API: **`federationSaved()`** and **`federationNotSaved()`**. A Save Failure Reason (discussed in 12.12.6.11) is passed as an argument to **`federationNotSaved()`**.

12.12.5.10 *Confirm Federation Restoration Request †* service

This service is mapped to the following two methods in the C++ API: **`requestFederationRestoreSucceeded()`** and **`requestFederationRestoreFailed()`**.

12.12.5.11 *Federate Restore Complete* service

This service is mapped to the following two methods in the C++ API: **`federateRestoreComplete()`** and **`federateRestoreNotComplete()`**.

12.12.5.12 *Federation Restored †* service

This service corresponds to two methods in the C++ API: **`federationRestored()`** and **`federationNotRestored()`**. A Restore Failure Reason (discussed in 12.12.6.12) is passed as an argument to **`federationNotRestored()`**.

12.12.5.13 *Connect* service

In addition to the supplied arguments listed with the description of this service, the C++ API requires the federate to supply an instance of the **`interface FederateAmbassador`**. The optional **`localSettingsDesignator`** parameter is required and can contain RTI implementation-specific configuration information. An empty string for the **`localSettingsDesignator`** parameter shall be used to indicate the default local settings.

In addition to the specified exceptions listed with the description of this service, the C++ API may also throw a **`CallNotAllowedFromWithinCallback`** exception. The **`CallNotAllowedFrom-WithinCallback`** exception indicates that this RTI ambassador service call cannot be made while the federate is processing a federate ambassador callback.

12.12.5.14 *Unpublish Object Class Attributes* service

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

12.12.5.15 *Unsubscribe Object Class Attributes* service

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

12.12.5.16 *Reserve Object Instance Name* service

The optional single name parameter or list of names parameter is implemented in the C++API as two methods: **`reserveObjectInstanceName()`** and **`reserveMultipleObjectInstance-Name()`**.

12.12.5.17 *Object Instance Name Reserved †* service

This service corresponds to two methods in the C++ API: **`objectInstanceNameReservation-Succeeded()`** and **`objectInstanceNameReservationFailed()`**.

12.12.5.18 *Multiple Object Instance Name Reserved †* service

This service corresponds to two methods in the C++ API: **`multipleObjectInstanceName-ReservationSucceeded()`** and **`multipleObjectInstanceNameReservationFailed()`**.

12.12.5.19 *Register Object Instance* service

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

12.12.5.20 *Discover Object Instance* service

The optional argument representing the producing joined federate designator is implemented in the C++ API as two **`discoverObjectInstance()`** methods: one that takes a **`FederateHandle`** as an argument for the producing joined federate designator, and one that does not.

12.12.5.21 *Update Attribute Values* service

The optional argument representing the timestamp is implemented in the C++ API as two **`uptdateAttributeValues()`** methods: one that takes a **`LogicalTime`** as an argument for the timestamp, and one that does not. The version that takes a **`LogicalTime`** also returns a **`MessageRetractionHandle`** for the message retraction designator.

12.12.5.22 *Reflect Attribute Values †* service

This service corresponds to three overloaded versions of the **`reflectAttributeValues()`** method in the C++ API. Which version the RTI calls is dependent on if the **`updateAttributeValues()`** was called with a timestamp and if the message was received TSO. See 12.8 for details how to determine which version the RTI will call.

12.12.5.23 *Send Interaction* service

The optional argument representing the timestamp is implemented in the C++ API as two **`sendInteraction()`** methods: one that takes a **`LogicalTime`** as an argument for the timestamp, and one that does not. The version that takes a **`LogicalTime`** also returns a **`MessageRetractionHandle`** for the message retraction designator.

12.12.5.24 *Receive Interaction †* service

This service corresponds to three overloaded versions of the **`receiveInteraction()`** method in the C++ API. Which version the RTI calls is dependent on if the **`sendInteraction()`** was called with a timestamp and if the message was received TSO. See 12.9 for details how to determine which version the RTI will call.

12.12.5.25 *Delete Object Instance* service

The optional argument representing the timestamp is implemented in the C++ API as two **`deleteObjectInstance()`** methods: one that takes a **`LogicalTime`** as an argument for the timestamp, and one that does not. The version that takes a **`LogicalTime`** also returns a **`MessageRetractionHandle`** for the message retraction designator.

12.12.5.26 *Remove Object Instance †* service

This service corresponds to three overloaded versions of the **`removeObjectInstance()`** method in the C++ API. See 12.10 for details on how to determine which version the RTI will call.

12.12.5.27 *Request Attribute Value Update* service

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

12.12.5.28 *Inform Attribute Ownership †* service

In the C++ API, 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.

12.12.5.29 *Register Object Instance With Regions* service

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

12.12.5.30 *Send Interaction With Regions* service

The optional argument representing the timestamp is implemented in the C++ API as two **`sendInteractionWithRegions()`** methods: one that takes a **`LogicalTime`** as an argument for the timestamp, and one that does not. The version that takes a **`LogicalTime`** also returns a **`MessageRetractionHandle`** for the message retraction designator.

12.12.5.31 *Resign Federation Execution* service

In addition to the specified exceptions listed with the description of this service, the C++ API may also throw a **`CallNotAllowedFromWithinCallback`** exception. This exception indicates that this RTI ambassador service call cannot be made while the federate is processing a federate ambassador callback.

12.12.5.32 *Disconnect* service

In addition to the specified exceptions listed with the description of this service, the C++ API may also throw a **`CallNotAllowedFromWithinCallback`** exception. This exception indicates that this RTI ambassador service call cannot be made while the federate is processing a federate ambassador callback.

12.12.5.33 *Evoke Callback* service

In addition to the specified exceptions listed with the description of this service, the C++ API may also throw a **`CallNotAllowedFromWithinCallback`** exception. This exception indicates that this RTI ambassador service call cannot be made while the federate is processing a federate ambassador callback.

12.12.5.34 *Evoke Multiple Callbacks* service

In addition to the specified exceptions listed with the description of this service, the C++ API may also throw a **`CallNotAllowedFromWithinCallback`** exception. This exception indicates that this RTI ambassador service call cannot be made while the federate is processing a federate ambassador callback.

12.12.6 Abstract datatypes

Subclause 1.4.2 presents general conventions used to describe the abstract datatypes used in Clause 4 through Clause 10. In 12.12.6.1 through 12.12.6.26, the abstract datatypes are further detailed by mapping an individual datatype to its implementation in the C++ API. Note that the datatypes from Clause 11 are explained in that clause and are not repeated here. Additionally, designators are a subject in their own right and are covered in 12.2.

The ISO/IEC committee ratified the C++ standard as ISO/IEC 14882 [B3]. The HLA C++ API utilizes standard C++ constructs to the fullest extent possible. For example, **`std::wstring`** is used to represent character strings that might previously have been represented with the C-style construct, pointer to **`char`** (i.e., **`char *`**). Additionally, several standard C++ containers are used to represent the sets, constrained sets of pairs, and collections described in Clause 3. These containers are **`std::set<T>`**, **`std::map<T>`**, and **`std::vector<T>`**. Also, the C++ standard **`std::auto_ptr<T>`** is used to implement strict memory ownership transfer when passing instances of dynamically allocated datatypes across API.

A description of the nature and properties of these standard C++ constructs is well beyond the scope of this clause. Instead, readers are referred to the C++ ISO standard as well as any of the vast assortment of C++ references. Two such excellent references are Austern [B1] and Stroustrup [B6].

12.12.6.1 Names, labels, and strings

The names, labels, and strings described in Clause 4 through Clause 10 of this specification are mapped to the **`wstring`** type.

12.12.6.2 Boolean values

Standard C++ provides a fundamental type, **`bool`**, intended for the representation of boolean values. This fundamental type is used for all boolean values in the C++ interface.

12.12.6.3 Exceptions

In keeping with the philosophy outlined in 12.12.6, exceptions in the C++ API have an interface identical with **`std::exception`**. In fact, all exceptions in the C++ API inherit from a class named **`Exception`**. This hierarchy allows all exceptions in the C++ API to be caught as a single base class.

12.12.6.4 Handles

Most designators in the C++ API are handles. Each handle is a distinct type, but they all share a common interface, defined by the **`DEFINE_HANDLE_CLASS`** macro and specialized into one of the following concrete classes: **`FederateHandle`**, **`ObjectClassHandle`**, **`InteractionClassHandle`**, **`ObjectInstanceHandle`**, **`AttributeHandle`**, **`ParameterHandle`**, **`DimensionHandle`**, **`MessageRetractionHandle`**, and **`RegionHandle`**.

Two instances of each class with identical types can be compared for equality or order (i.e., less than), and one can be assigned to the other. A copy constructor is provided as well. These handles are always generated by an RTI implementation; however, a default constructor is provided for convenience. Additionally, an **`encode()`** method is provided that returns an encoded form of a handle, **`VariableLengthData`**.

`VariableLengthData` has a **`data()`** method that returns an untyped pointer (i.e., a **`void const *`**) as well as a **`size()`** method that returns the size in bytes (represented as a **`unsigned long`**) of the data returned by the **`data()`** method. A copy constructor and a copy-assignment operator are also provided.

Using this interface, it is possible for the RTI user to save and restore any handle provided by the RTI as well as to pass handles as attribute values or parameter values among federates. To send a handle as an attribute value or as a parameter value, the **`encode()`** method of the handle should be invoked by the federate programmer. The returned **`VariableLengthData`** class has a **`data()`** method and a **`size()`** method that can be used to fill in the attribute value or parameter value. Once delivered, the encoded handle can be reconstituted into an instance of its corresponding handle class using that handle class's constructor.

Note well that the pointer value returned by the **`data()`** method is not required to have any particular byte alignment. Consequently, naively attempting to simply type cast this pointer and assign its contents to some type may generate an exception in the processor's memory management unit and likely result in the crash of the program. If for some reason it is desirable to perform this type of operation, a function such as **`memcpy()`** should be used.

The **`VariableLengthData`** class has a constructor that requires the user to provide an untyped pointer (i.e., a **`void const *`**) as well as the size in bytes (represented as a **`unsigned long`**) of the data to which it points. Methods to return these values are provided as well as is a copy constructor and a copy-assignment operator.

Handle classes also provide a **`toString()`** 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, employ the appropriate support service from Clause 10.

12.12.6.5 Resignation directive

The *Resign Federation Execution* service allows the specification of one of six directives. To provide this specification, a **`ResignAction`** enumeration is defined. The six different directives of *Resign Federation Execution* are defined as the six enumerated values that instances of the **`ResignAction`** enumeration may take. The **`ResignAction`** enumeration has the following values:

- **UNCONDITIONALLY_DIVEST_ATTRIBUTES**
- **DELETE_OBJECTS**
- **CANCEL_PENDING_OWNERSHIP_ACQUISITIONS**
- **DELETE_OBJECTS_THEN_DIVEST**
- **CANCEL_THEN_DELETE_THEN_DIVEST**
- **NO_ACTION**

12.12.6.6 User-supplied tag

Several services in the HLA allow the user to supply additional information to be transported by the RTI in an opaque manner, such as the *Update Attribute Values* service. The C++ API uses the **`VariableLengthData`** class to achieve this goal.

12.12.6.7 Sets of designators

Sets of designators are implemented using the standard C++ set container with the appropriate handle type. For convenience, each different set of designators has been aliased using a **`typedef`**. To ensure maximum portability between RTI implementations, federate programmers should use only this **`typedef`**. The list of set types with their corresponding **`typedef`**s are as follows:

- A set of federate designators uses **`std::set<FederateHandle>`**, and the **`typedef FederateHandleSet`** should be used.
- A set of attribute designators uses **`std::set<AttributeHandle>`**, and the **`typedef AttributeHandleSet`** should be used.
- A set of attribute designators uses **`std::set<ParameterHandle>`**, and the **`typedef ParameterHandleSet`** should be used.
- A set of dimension designators uses **`std::set<DimensionHandle>`**, and the **`typedef DimensionHandleSet`** should be used.
- A set of region designators uses **`std::set<RegionHandle>`**, and the **`typedef RegionHandleSet`** should be used.

12.12.6.8 Success indicators

Several of the requests a federate may make of an RTI shall be acknowledged by the RTI to the federate. This acknowledgment can indicate that the request succeeded or failed. In the C++ API, each request acknowledgment corresponds to a positive and a negative callback method on the **`FederateAmbassador`**, e.g., **`federationSaved()`** and **`federationNotSaved()`**, **`requestFederationRestoreSucceeded()`** and **`requestFederationRestoreFailed()`**, and **`federationRestored()`** and **`federationNotRestored()`**.

12.12.6.9 Callback model

When the **`connect()`** method is invoked by a federate, it must supply an instance of the **`CallbackModel`** enumeration as an argument. The **`CallbackModel`** enumeration provides the following values:

— **HLA_IMMEDIATE**
— **HLA_EVOKED**

12.12.6.10 Synchronization failure reason

When the **`synchronizationPointRegistrationFailed()`** method is invoked by the RTI on a federate, the RTI provides an instance of a **`SynchronizationFailureReason`** enumeration as an argument. The **`SynchronizationFailureReason`** enumeration provides the following values:

— **SYNCHRONIZATION_POINT_LABEL_NOT_UNIQUE**
— **SYNCHRONIZATION_SET_MEMBER_NOT_JOINED**

12.12.6.11 Save Failure Reason

When the **`federationNotSaved()`** method is invoked by the RTI on a federate, the RTI provides an instance of a **`SaveFailureReason`** enumeration as an argument. The **`SaveFailureReason`** enumeration provides the following values:

— **UNABLE_TO_SAVE**
— **FEDERATE_REPORTED_FAILURE_DURING_SAVE**
— **FEDERATE_RESIGNED_DURING_SAVE**
— **RTI_DETECTED_FAILURE_DURING_SAVE**
— **SAVE_TIME_CANNOT_BE_HONORED**
— **SAVE_ABORTED**

12.12.6.12 Restore Failure Reason

When the **`federationNotRestored()`** method is invoked by the RTI on a federate, the RTI provides an instance of a **`RestoreFailureReason`** enumeration as an argument. The **`RestoreFailureReason`** enumeration provides the following values:

— **RTI_UNABLE_TO_RESTORE**
— **FEDERATE_REPORTED_FAILURE_DURING_RESTORE**
— **FEDERATE_RESIGNED_DURING_RESTORE**
— **RTI_DETECTED_FAILURE_DURING_RESTORE**
— **RESTORE_ABORTED**

12.12.6.13 List of joined federates save status

The list of joined federates and the save status of each is implemented as a vector of federate designator and save status pairs. In the C++ API, this corresponds to an **`vector<pair< FederateHandle, SaveStatus>>`**. For convenience, this type has been aliased using a **`typedef`** to the name **`FederateHandleSaveStatusPairVector`**. To ensure maximum portability between RTI implementations, federate programmers should use only this **`typedef`**.

`SaveStatus` is an enumeration that provides the following values:

— **NO_SAVE_IN_PROGRESS**

— **FEDERATE_INSTRUCTED_TO_SAVE**
— **FEDERATE_SAVING**
— **FEDERATE_WAITING_FOR_FEDERATION_TO_SAVE**

12.12.6.14 List of joined federates restore status

The list of joined federates and the restore status of each is implemented as a vector of federate designator and restore status pairs. In the C++ API, this corresponds to an **`vector<pair< FederateHandle, RestoreStatus>>`**. For convenience, this type has been aliased using a **`typedef`** to the name **`Federate HandleRestoreStatusPairVector`**. To ensure maximum portability between RTI implementations, federate programmers should use only this **`typedef`**.

`RestoreStatus` is an enumeration that provides the following values:

— **NO_RESTORE_IN_PROGRESS**
— **FEDERATE_RESTORE_REQUEST_PENDING**
— **FEDERATE_WAITING_FOR_RESTORE_TO_BEGIN**
— **FEDERATE_PREPARED_TO_RESTORE**
— **FEDERATE_RESTORING**
— **FEDERATE_WAITING_FOR_FEDERATION_TO_RESTORE**

12.12.6.15 Passive subscription indicator

The *Subscribe Object Class Attributes*, *Subscribe Interaction Class*, *Subscribe Object Class Attributes With Regions*, and *Subscribe Interaction Class With Regions* services each may optionally provide an **`bool`** value to indicate whether the subscription is active (**`true`** means the subscription is active). In the C++ API, this is a default parameter to the methods corresponding to each of the services. The default value is **`true`**.

12.12.6.16 Constrained set of attribute designator and value pairs

The *Update Attribute Values* and *Reflect Attribute Values †* services require a constrained set of attribute designator and value pairs as arguments. A constrained set of attribute designator and value pairs is implemented in the C++ API as an **`map<AttributeHandle, VariableLengthData>`**. For convenience, this type has been aliased using a **`typedef`** to the name **`AttributeHandleValueMap`**. To ensure maximum portability between RTI implementations, federate programmers should use only this **`typedef`**.

12.12.6.17 Constrained set of interaction parameter designator and value pairs

The *Send Interaction*, *Receive Interaction †*, and *Send Interaction With Regions* services require a constrained set of parameter designator and value pairs as arguments. A constrained set of parameter designator and value pairs is implemented in the C++ API as an **`map<ParameterHandle, VariableLengthData>`**. For convenience, this type has been aliased using a **`typedef`** to the name **`ParameterHandleValueMap`**. To ensure maximum portability between RTI implementations, federate programmers should use only this **`typedef`**.

12.12.6.18 Message order type

Message order types are represented an enumeration named **`OrderType`**, which has the following values: **`RECEIVE`** and **`TIMESTAMP`**. These values can be passed to the RTI or compared to an instance of an **`OrderType`** object returned by the RTI.

12.12.6.19 Transportation type

As with message order types described above, transportation types are represented by an enumeration. An RTI shall provide the following enumerations: **`RTI_RELIABLE`** and **`RTI_BEST_EFFORT`**. These enumerations may be passed to the RTI or compared to an instance of a **`TransportationType`** enumeration returned by the RTI.

It is possible that a FDD may contain additional transportation types. In this case, the RTI implementation shall extend the **`TransportationType`** enumerated values to provide a unique value for each new transportation type. These new transportation type enumerated values shall be retrievable from the RTI via the **`getTransportationType`** service. In no case shall the existing **`TransportationType`** enumerations or the Enums.h header file be modified or eliminated.

12.12.6.20 Ownership designator

The *Inform Attribute Ownership †* service supplies an ownership designator as one of its arguments. In the C++ API, this corresponds to three distinct methods: **`attributeIsNotOwned()`**, **`attributeIsOwnedByRTI()`**, and **`informAttributeOwnership()`**. All three methods take an **`ObjectInstanceHandle`** and an **`AttributeHandle`** as arguments. In addition, **`informAttributeOwnership()`** takes an **`FederateHandle`** as an argument. Together, these three methods combine to represent the ownership designator of the *Inform Attribute Ownership †* service.

12.12.6.21 Definition indicator

The *Query GALT* and *Query LITS* services are required to return an indicator to define whether their return value is valid. Since both services return values of type **`auto_ptr<LogicalTime>`**, the definition indicator is trivial: if the returned value is **`NULL`**, the value is undefined (invalid); otherwise, the returned value is defined (valid).

12.12.6.22 Optional message retraction designator

The *Update Attribute Values*, *Send Interaction*, *Send Interaction with Regions*, and *Delete Object Instance* services may return a message retraction designator under certain circumstances. In the C++ API, this is implemented much like the definition indicator described above. The methods corresponding to these services always return an **`auto_ptr<MessageRetractionHandle>`**. If the returned value is **`NULL`**, then the operation did not generate a message retraction designator; otherwise, the **`auto_ptr<MessageRetractionHandle>`** holds the returned message retraction designator.

12.12.6.23 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. In the C++ API, this corresponds to a **`vector< pair<AttributeHandleSet, RegionHandleSet>>`**. For convenience, this type has been aliased using a **`typedef`** to the name **`AttributeHandleSetRegionHandleSetPair Vector`**. To ensure maximum portability between RTI implementations, federate programmers should use only this **`typedef`**.

12.12.6.24 Logical time, timestamps, and lookahead

The C++ API defines the logical time-related classes as abstract base classes. The method for obtaining a printable string representation of a **`LogicalTime`** and a **`LogicalTimeInterval`** is **`toString()`** and it returns a **`wstring.`** The **`encode()`** and **`decode()`** methods defined for the **`LogicalTime`** and **`LogicalTimeInterval`** classes use objects of type **`VariableLengthDataType`** for the variable array. The C++ API also provides **`encode()`** and **`decode()`** methods that allow encoding and decoding

directly into a **void** * buffer. The encoding and decoding methods are available on the **LogicalTime**, **LogicalTimeInterval**, and **LogicalTimeFactory** classes.

In addition to the methods mentioned in 12.3, the C++ **LogicalTime** interface provides the following methods:

— **setInitial()** that sets the current instance to the initial value
— **setFinal()** that sets the current instance to the final value
— **encodedLength()** that returns the number of bytes needed to encode the current instance
— **implementationName()** that returns a **std::wstring** with the name of the logical time implementation that is in use by the **createFederationExecution** method

In addition to the methods mentioned in 12.3, the C++ **LogicalTimeInterval** interface provides the following methods:

— **setZero()** that sets the current instance to the zero value
— **setEpsilon()** that sets the current instance to the epsilon value
— **encodedLength()** that returns the number of bytes needed to encode the current instance
— **implementationName()** that returns a **std::wstring** with the name of the logical time implementation that is in use by the **createFederationExecution** method

In addition to the methods mentioned in 12.3, the C++ **LogicalTimeFactory** interface provides the following methods:

— **setZero()** that sets the current instance to the zero value
— **makeInitial()** to create an instance of a **LogicalTime** with the initial value
— **makeFinal()** to create an instance of a **LogicalTime** with the final value
— **makeZero()** to create an instance of a **LogicalTimeInterval** with the value of zero
— **makeEpsilon()** to create an instance of a **LogicalTimeInterval** with the value of epsilon

A C++ library that implements the logical time classes shall supply a exported static method with the following signature that allows an RTI implementation to create an instance of the **LogicalTimeFactory()**. Federates should not use this interface to obtain an **LogicalTimeFactory** but should instead use the **getTimeFactory()** method on the RTI ambassador.

```
static std::auto_ptr< LogicalTimeFactory > makeLogicalTimeFactory(std::wstring const & implementationName);
```

12.12.6.24.1 Standardized time types

The implementations of the standardized time types shall be placed in the **hla.rti1516e** namespace and have the names specified in Table 23.

Table 23—C++ API provided time implementations

Property	64-bit integer time	64-bit float time
Symbolic name	“HLAinteger64Time”	“HLAfloat64Time”
C++ time class name	rti1516e::HLAinteger64Time	rti1516e::HLAfloat64Time
C++ interval class name	rti1516e::HLAinteger64Interval	rti1516e::HLAfloat64Interval
C++ time factory class name	rti1516e::HLAinteger64TimeFactory	rti1516e::HLAfloat64TimeFactory

12.12.6.25 Dimension upper bound and range lower and upper bounds

Several DDM-related services allow bounds to be gotten and set. The *Get Dimension Upper Bound* service gets the upper bound of a particular dimension. The remaining services are used to get and set the bounds of ranges of a region. All bounds are represented in the C++ API using an **`unsigned`** integer.

Range lower and upper bounds are gathered into a **`RangeBounds`** object.

12.12.6.26 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 in the C++ API using a **`double`**.

12.13 Web Service Definition Language (WSDL)

12.13.1 Overview

12.13.1.1 Standards and terminology

The Web Service API is specified using the WSDL version 1.1. WSDL uses XML Schema and is based on the SOAP 1.1 specification that in turn uses XML version 1.0. The design of the Web Service API follows the guidelines of the Web Service Interoperability (WS-I) standard version 1.1. Full references to the above standards are shown in Table 24.

Table 24—WSDL referenced standards

Specification	Location
WSDL 1.1	http://www.w3.org/TR/wsdl.html
SOAP 1.1	http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
XML 1.0	http://www.w3.org/TR/REC-xml/
HTTP 1.1	http://www.ietf.org/rfc/rfc2616
XML Schema Part 1: Structures	http://www.w3.org/TR/xmlschema-1/
XML Schema Part 2: Datatypes	http://www.w3.org/TR/xmlschema-2/
WS-I Basic Profile 1.1	http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html

The following terms in the text are used based on their WSDL and Web Service sense unless otherwise indicated: service, consumer, provider, nillable, port, transportation and http.

The namespace http://sisostds.org/hla/evolved/wsdl/v1 is used for this WSDL.

12.13.1.2 HLA Web service consumers and providers

This API differs from the C++ and Java APIs since it is not actually a programming API but a protocol used for communicating between a federate and the RTI. The federate in this context is considered to be a Web Service Consumer. The Web Service is provided by a software component called the Web Service Provider RTI Component (WSPRC), often referred to as the Provider.

Federates that wish to interoperate must connect to RTI Components [Local Runtime Components (LRCs) or WSPRCs] that are able to establish a common federation execution. No additional restrictions are imposed on the topology or implementation of the RTI. One RTI may provide several WSPRCs. One WSPRC may support several simultaneously connected federates. One WSPRC may provide several ports that Consumers can connect to.

12.13.1.3 Implementation language considerations

The Web Service API covers the full HLA interface specification. A Web Service federate or RTI may be implemented in any language and environment capable of Web Service communication.

There are several ways to implement a federate using the WSDL API. It is, for example, possible to generate code for several languages using WSDL-to-source code generators. Note that if code is generated for C++ and Java, this code is likely to differ from the corresponding standard HLA APIs.

12.13.1.4 Transportation protocols and ports

The HLA Web Service API shall make use of Hypertext Transfer Protocol (HTTP) for transportation.

The WSDL API in Annex D uses a placeholder for the port, which is the connection point for a Consumer. When implementing the WSDL API, this needs to be replaced with the port where the HLA Web Service Provider is available.

12.13.2 Services

12.13.2.1 Optional parameters and polymorphism

Parameters that are described as optional in this standard are defined as Nillable in the Web Service API.

In case of polymorphism where the same HLA service can be called with parameters of different types, these HLA services are divided into several Web services.

12.13.2.2 Exceptions

All exceptions are implemented as SOAP Fault elements and are handled according to the SOAP standard see 12.13.1.1). One SOAP Fault element is generated for each exception.

12.13.2.3 FDD and connection parameters

When invoking the *Create Federation Execution* service, the FDD shall be referenced by a set of URLs. The URLs in this set shall point to the FOM modules that apply to that FDD. These FOM modules shall be accessible for the WSPRC.

The local settings designator in the connect call shall be used to provide information to the WSPRC to be used for connecting to other RTI components.

12.13.2.4 Modified services

The *Request Attribute Value Update* service may take either an object instance or a class designator as its first parameter. This service is implemented as two different services in the WSDL API:

— Request Object Instance Attribute Value Update
— Request Object Class Attribute Value Update

The *Inform Attribute Ownership* service (callback) is implemented using the following calls:

— Inform Attribute Ownership
— Attribute Is Not Owned
— Attribute Owned By RTI

The *Federate Save Complete* service is implemented using the following calls:

— Federate Save Complete
— Federate Save Not Complete

The *Federate Restore Complete* service is implemented using the following calls:

— Federate Restore Complete
— Federate Restore Not Complete

12.13.3 Datatypes

12.13.3.1 Encoding of parameters in the interface specification

Table 25 summarizes how parameters shall be represented in the Web Service API.

The xsd prefix indicates the http://www.w3.org/2001/XMLSchema namespace.

Table 25—Parameter type representations

Parameter	Representation in Web Service API
userSuppliedTag	xsd:base64Binary
timeStamp	xsd:base64Binary
logicalTime	xsd:base64Binary
lookahead	xsd:base64Binary
messageRetractionDesignator	xsd:base64Binary
regionDesignator	xsd:base64Binary
attributeValue	xsd:base64Binary
parameterValue	xsd:base64Binary
encodedHandle	xsd:base64Binary
passiveSubscriptionIndicator	xsd:boolean
requestSuccessIndicator	xsd:boolean
federationRestoreSuccessIndicator	xsd:boolean
reservationSuccessIndicator	xsd:boolean
registrationSuccessIndicator	xsd:boolean
federationSaveSuccessIndicator	xsd:boolean
instanceAttributeOwnershipIndicator	xsd:boolean
timeIsValid	xsd:boolean

Table 25—Parameter type representations *(continued)*

Parameter	Representation in Web Service API
dimensionBound	xsd:long
rangeBound	xsd:long
normalizedValue	xsd:long
maximumCallbackCount	xsd:long
pendingIndicator	xsd:long
updateRate	xsd:double
federateName	xsd:string
federationExecutionName	xsd:string
objectInstanceName	xsd:string
objectClassName	xsd:string
attributeName	xsd:string
interactionClassName	xsd:string
parameterName	xsd:string
dimensionName	xsd:string
transportationName	xsd:string
orderName	xsd:string
localSettingsDesignator	xsd:string
federateDesignator	xsd:string
objectClassDesignator	xsd:string
interactionClassDesignator	xsd:string
objectInstanceDesignator	xsd:string
updateRateDesignator	xsd:string
parameterDesignator	xsd:string
attributeDesignator	xsd:string
dimensionDesignator	xsd:string
ownershipDesignator	xsd:string
federateType	xsd:string
synchronizationPointLabel	xsd:string
federationSaveLabel	xsd:string
faultDescription	xsd:string
fomDocumentDesignator	xsd:anyURI
AttributeDesignatorValuePair	<xsd:sequence> of attributeDesignator and attributeValue
ParameterValuePair	<xsd:sequence> of parameterDesignator and parameterValue

Table 25—Parameter type representations *(continued)*

Parameter	Representation in Web Service API
AttributeDesignatorRegionDesignatorPair	<xsd:sequence> of attributeDesignator and regionDesignator
FederateDesignatorSaveStatusPair	<xsd:sequence> of federateDesignator and SaveStatus
FederateDesignatorRestoreStatusPair	<xsd:sequence> of federateDesignator and RestoreStatus
TimeQueryReturn	<xsd:sequence> of timeIsValid and logicalTime
RangeBounds	<xsd:sequence> of two rangeBound
Callback	<xsd:choice> between one of the callbacks
ResignAction	<xsd:enumeration> of <xsd:string>
OrderType	<xsd:enumeration> of <xsd:string>
TransportationType	<xsd:enumeration> of <xsd:string>
ServiceGroup	<xsd:enumeration> of <xsd:string>
SaveFailureReason	<xsd:enumeration> of <xsd:string>
SynchronizationPointFailureReason	<xsd:enumeration> of <xsd:string>
RestoreFailureReason	<xsd:enumeration> of <xsd:string>
SaveStatus	<xsd:enumeration> of <xsd:string>
RestoreStatus	<xsd:enumeration> of <xsd:string>
AttributeDesignatorSet	<xsd:sequence> of attributeDesignator
AttributeDesignatorValuePairSet	<xsd:sequence> of AttributeDesignatorValuePairs
ParameterDesignatorValuePairSet	<xsd:sequence>of ParameterDesignatorValuePairs
DimensionDesignatorSet	<xsd:sequence>of dimensionDesignators
RegionDesignatorSet	<xsd:sequence> of regionDesignators
FederateDesignatorSet	<xsd:sequence> of federateDesignators
AttributeDesignatorRegionDesignatorPairList	<xsd:sequence> of AttributeDesignatorRegionDesignatorPairs
CallbackArray	<xsd:sequence> of Callbacks
CallbackReturn	<xsd:sequence> of CallbackArray and pendingIndicator
FederateDesignatorSaveStatusPairList	<xsd:sequence> of FederateDesignatorSaveStatusPairs
FederateDesignatorRestoreStatusPairList	<xsd:sequence> of FederateDesignatorRestoreStatusPairs
FederationExecutionNameSet	<xsd:sequence> of federationExecutionNames

12.13.3.2 Encoding of FDD-defined data values

Certain data shall first be encoded according to the HLA OMT and then encoded using xsd:base64Binary before being sent. This requirement applies to the following data:

— Attribute values
— Parameter values
— Logical timestamps

Web Service federates have the option of using the encoding helpers available in the Java and C++ APIs for the encoding. In addition to this option, a time-regulating and/or time-constrained federate needs access to a time class implementation that is compatible with the one used in the federation.

12.13.3.3 Time representation and implementation

The WSDL API requires no standardized local RTI component for a federate; in other words, no local time or interval classes are provided. The time values are exchanged between the federate and the RTI as byte arrays. An RTI shall support federates using the WSDL API that use the standardized support services below for encoding, decoding, and updating timestamps as well as federates performing the corresponding operations without using these services.

Implementers of the WSDL API shall provide, as part of their implementation, implementations of encoding and decoding services for two standardized logical time types: **`HLAinteger64Time`** and **`HLAfloat64Time`** using the service names shown in Table 26.

Table 26—WSDL standard time types

Property	64-bit integer time service name	64-bit float time service name
Symbolic name	HLAinteger64Time	HLAfloat64Time
Encode time value	encodeHLAinteger64Time	encodeHLAfloat64Time
Decode time value	decodeHLAinteger64Time	decodeHLAfloat64Time
Encode interval value	encodeHLAinteger64Interval	encodeHLAfloat64Interval
Decode interval value	decodeHLAinteger64Interval	decodeHLAfloat64Interval

12.13.4 Handles

12.13.4.1 Handles for FDD data

Unlike the Java and C++ APIs, the WSDL API does not use handles for FDD data. Instead, the WSDL API uses fully qualified names as designated in Table 27.

Table 27—Handle name sources

Handle	Source for name
Object Class Handle	fully qualified name from FDD object class structure table
Attribute Handle	fully qualified name from FDD attribute table
Interaction Class Handle	fully qualified name from FDD interaction class structure table
Parameter Handle	fully qualified name from FDD parameter table
Dimension Handle	name from FDD dimension table
Object Instance Handle	object instance name as returned from registerObjectInstance operation

Table 27—Handle name sources *(continued)*

Handle	Source for name
Federate Handle	federate name as provided to the joinFederationExecution operation
Transportation Type	name from FDD transportation type table
Order Type	name from FDD attribute table or FDD parameter table
Update Rate	name from FDD update rate table

To simplify porting between the WSDL API and other APIs, the support services for FDD data handles are still available. Services that return handles in the Java or C++ API shall instead return the fully qualified name in the WSDL API. Services that take handles as input in the Java or C++ API shall instead take the fully qualified name as input in the WSDL API. Exceptions for incorrect values shall be thrown as for the C++ and Java API. The following services take a handle or fully qualified name, respectively:

— Get Object Class Handle / Get Object Class Name
— Get Attribute Handle / Get Attribute Name
— Get Interaction Class Handle / Get Interaction Class Name
— Get Parameter Handle / Get Parameter Name
— Get Dimension Handle / Get Dimension Name
— Get Object Instance Handle / Get Object Instance Name
— Get Federate Handle / Get Federate Name

12.13.4.2 Services for encoding and decoding handles

The WSDL API still offers the ability to exchange handles with other federates. The principle is the same as for the C++ and Java API: a handle must be encoded prior to sending and decoded by the receiver. The services shown in Table 28 are provided to support this activity.

Table 28—Handle encoding and decoding services

Handle type	Encoding service	Decoding service
Federate Handle	*Encode Federate Handle*	*Decode Federate Handle*
Object Class Handle	*Encode Object Class Handle*	*Decode Object Class Handle*
Interaction Class Handle	*Encode Interaction Class Handle*	*Decode Interaction Class Handle*
Object Instance Handle	*Encode Object Instance Handle*	*Decode Object Instance Handle*
Parameter Handle	*Encode Parameter Handle*	*Decode Parameter Handle*
Attribute Handle	*Encode Attribute Handle*	*Decode Attribute Handle*
Dimension Handle	*Encode Dimension Handle*	*Decode Dimension Handle*

12.13.5 Special Web Service considerations

12.13.5.1 Delivery of callbacks

The current WSDL API supports only requests from the Consumer (federate) and responses from the Provider (WSPRC). To get callbacks from the WSPRC to the Consumer, the following service needs to be called:

— EvokeMultipleCallbacks (Maximum Callback Count).

The return value of this service is a CallbackArray element containing a `<xsd:sequence>` of callback elements as specified in the WSDL. A maximum callback count is supplied. The minimum and maximum time is removed. The RTI shall return no more than the specified number of callbacks.

The pending indicator is returned in both cases to provide the number of pending callbacks.

Note: The Evoke Callback function is unavailable in the WSDL API since the call EvokeMultipleCallbacks does the same function.

How frequently these services need to be called may vary between different federates, federations, and scenarios. The Consumer may also need to take the session timeout (described below) into consideration.

12.13.5.2 Session management

An ongoing connection between a federate and the RTI in the C++ and Java API is constituted by a federate ambassador and RTI ambassador pair. In the WSDL API, an ongoing connection is instead implemented through a session identifier (ID). The session ID is initially provided by the WSPRC when the Consumer issues the connect call unless it already has a valid session ID. This session ID is subsequently used to identify this ongoing connection between the WSPRC and the Consumer. The session ID is sent as a SOAP-header in requests and responses.

The WSPRC (RTI) may terminate the session and dispose of the session data if the Consumer has not called the WSPRC for a certain time. The session is then considered unavailable. This situation may, for example, happen if the Consumer has terminated unexpectedly.

There is no prescribed value for this timeout value; in other words, different schemas may be implemented including an infinite timeout value.

A session can also become unavailable for other reasons, for example, if the WSPRC fails and is restarted.

12.13.5.3 Fault tolerance

Fault tolerance considerations must take into account that the federate may encounter problems with the connection to the WSPRC. The WSPRC may also encounter problems with the connection to the federation.

In case the WSPRC loses the connection to the federation, the *Connection Lost †* call shall be sent to the Consumer. Any call but the *Evoke Multiple Callbacks* service shall raise the “Not Connected” exception after that.

12.13.5.3.1 Session timeout

Losing the session between the Consumer and the WSPRC corresponds to losing the reference to the RTI ambassador in a federate using the C++ or Java API.

In case the session between the federate and the WSPRC has become unavailable, all calls to the WSPRC except for connect shall raise the “Not Connected” exception.

In case a Consumers Web Service call to the WSPRC times out, the Consumer may act on its own choosing. It may try to connect later using the session ID already provided. Ultimately it may try to establish a new connection to the same or another WSPRC.

13. Conformance

The interface defined in this document between a federate and an RTI places requirements on both sides of the interface. This clause defines the measures for assessment of federates and RTIs, based upon the normative clauses of this specification. Because a large number of federates rely on a relatively much smaller number of RTIs, the measures are much more strict for RTIs. Within the HLA community, verification capabilities will conduct these assessments experimentally and publicly certify the results. Only when results certified by a recognized authority are publicly available shall the label “HLA Conforming” be used.

13.1 Federate conformance

An HLA-conforming federate shall document the elements of the interface defined in this document that are used in the conformance table of its SOM as specified in IEEE Std 1516.2-2010.

13.2 RTI conformance

An HLA-conforming RTI shall implement all the joined federate-initiated services and invoke all the RTI-initiated services as defined in this document. Mapping these interfaces to programming languages shall be done in accordance with Clause 12, and at least one language-specific API shall be implemented. Conformance verification shall be done according to the language API(s) implemented, and certified results shall reflect the language API(s) verified.

Annex A

(informative)

API information

C++ and Java are such common programming languages that this standard dictates a language binding for each API (see Annex B and Annex C) that must be used in order to comply with this standard. The same applies to the WSDL API (see Annex D). However, for other languages, e.g., Ada 95, C#, Python, it is still possible to develop RTI APIs that may be considered compliant. It is up to the implementer of the particular API to determine how to map the body of this standard to the respective programming language.

Annex B

(normative)

Java API

The files listed below shall comprise the HLA Java API. These files are grouped in a file named IEEE1516-2010_Java_API.zip, which is part of this specification and can be downloaded from either of the following locations:

— http://standards.ieee.org/downloads/1516/1516.1-2010/IEEE1516-2010_Java_API.zip
— http://www.sisostds.org/apis.aspx/IEEE1516-2010_Java_API.zip

hla:
rti1516e/

hla/rti1516e:
AttributeHandle.java
AttributeHandleFactory.java
AttributeHandleSet.java
AttributeHandleSetFactory.java
AttributeHandleValueMap.java
AttributeHandleValueMapFactory.java
AttributeRegionAssociation.java
AttributeSetRegionSetPairList.java
AttributeSetRegionSetPairListFactory.java
CallbackModel.java
DimensionHandle.java
DimensionHandleFactory.java
DimensionHandleSet.java
DimensionHandleSetFactory.java
FederateAmbassador.java
FederateHandle.java
FederateHandleFactory.java
FederateHandleSaveStatusPair.java
FederateHandleSet.java
FederateHandleSetFactory.java
FederateRestoreStatus.java
FederationExecutionInformation.java
FederationExecutionInformationSet.java
InteractionClassHandle.java
InteractionClassHandleFactory.java
LogicalTime.java
LogicalTimeFactory.java
LogicalTimeFactoryFactory.java
LogicalTimeInterval.java
MessageRetractionHandle.java
MessageRetractionReturn.java
NullFederateAmbassador.java
ObjectClassHandle.java
ObjectClassHandleFactory.java

ObjectInstanceHandle.java
ObjectInstanceHandleFactory.java
OrderType.java
ParameterHandle.java
ParameterHandleFactory.java
ParameterHandleValueMap.java
ParameterHandleValueMapFactory.java
RTIambassador.java
RangeBounds.java
RegionHandle.java
RegionHandleSet.java
RegionHandleSetFactory.java
ResignAction.java
RestoreFailureReason.java
RestoreStatus.java
RtiFactory.java
RtiFactoryFactory.java
SaveFailureReason.java
SaveStatus.java
ServiceGroup.java
SynchronizationPointFailureReason.java
TimeQueryReturn.java
TransportationTypeHandle.java
TransportationTypeHandleFactory.java
encoding/
exceptions/
time/

hla/rti1516e/encoding:
ByteWrapper.java
DataElement.java
DataElementFactory.java
EncoderFactory.java
HLAASCIIchar.java
HLAASCIIstring.java
HLAboolean.java
HLAbyte.java
HLAfixedArray.java
HLAfixedRecord.java
HLAfloat32BE.java
HLAfloat32LE.java
HLAfloat64BE.java
HLAfloat64LE.java
HLAhandle.java
HLAinteger16BE.java
HLAinteger16LE.java
HLAinteger32BE.java
HLAinteger32LE.java
HLAinteger64BE.java
HLAinteger64LE.java
HLAlogicalTime.java
HLAlogicalTimeInterval.java
HLAoctet.java
HLAoctetPairBE.java

HLAoctetPairLE.java
HLAopaqueData.java
HLAunicodeChar.java
HLAunicodeString.java
HLAvariableArray.java
HLAvariantRecord.java

hla/rti1516e/exceptions:
AlreadyConnected.java
AsynchronousDeliveryAlreadyDisabled.java
AsynchronousDeliveryAlreadyEnabled.java
AttributeAcquisitionWasNotCanceled.java
AttributeAcquisitionWasNotRequested.java
AttributeAlreadyBeingAcquired.java
AttributeAlreadyBeingChanged.java
AttributeAlreadyBeingDivested.java
AttributeAlreadyOwned.java
AttributeDivestitureWasNotRequested.java
AttributeNotDefined.java
AttributeNotOwned.java
AttributeNotPublished.java
AttributeNotRecognized.java
AttributeNotSubscribed.java
AttributeRelevanceAdvisorySwitchIsOff.java
AttributeRelevanceAdvisorySwitchIsOn.java
AttributeScopeAdvisorySwitchIsOff.java
AttributeScopeAdvisorySwitchIsOn.java
CallNotAllowedFromWithinCallback.java
ConnectionFailed.java
CouldNotCreateLogicalTimeFactory.java
CouldNotDecode.java
CouldNotDiscover.java
CouldNotInitiateRestore.java
CouldNotOpenFDD.java
DeletePrivilegeNotHeld.java
ErrorReadingFDD.java
FederateAlreadyExecutionMember.java
FederateHandleNotKnown.java
FederateHasNotBegunSave.java
FederateInternalError.java
FederateNameAlreadyInUse.java
FederateNotExecutionMember.java
FederateOwnsAttributes.java
FederateServiceInvocationsAreBeingReportedViaMOM.java
FederateUnableToUseTime.java
FederatesCurrentlyJoined.java
FederationExecutionAlreadyExists.java
FederationExecutionDoesNotExist.java
IllegalName.java
IllegalTimeArithmetic.java
InTimeAdvancingState.java
InteractionClassAlreadyBeingChanged.java
InteractionClassNotDefined.java
InteractionClassNotPublished.java

InteractionClassNotRecognized.java
InteractionClassNotSubscribed.java
InteractionParameterNotDefined.java
InteractionParameterNotRecognized.java
InteractionRelevanceAdvisorySwitchIsOff.java
InteractionRelevanceAdvisorySwitchIsOn.java
InvalidAttributeHandle.java
InvalidDimensionHandle.java
InvalidFederateHandle.java
InvalidInteractionClassHandle.java
InvalidLocalSettingsDesignator.java
InvalidLogicalTime.java
InvalidLogicalTimeInterval.java
InvalidLookahead.java
InvalidMessageRetractionHandle.java
InvalidObjectClassHandle.java
InvalidOrderName.java
InvalidOrderType.java
InvalidParameterHandle.java
InvalidRangeBound.java
InvalidRegion.java
InvalidRegionContext.java
InvalidResignAction.java
InvalidServiceGroup.java
InvalidTransportationName.java
InvalidTransportationTypeHandle.java
InvalidUpdateRateDesignator.java
JoinedFederateIsNotInTimeAdvancingState.java
LogicalTimeAlreadyPassed.java
MessageCanNoLongerBeRetracted.java
NameNotFound.java
NoAcquisitionPending.java
NoRequestToEnableTimeConstrainedWasPending.java
NoRequestToEnableTimeRegulationWasPending.java
NotConnected.java
ObjectClassNotDefined.java
ObjectClassNotPublished.java
ObjectClassNotRecognized.java
ObjectClassRelevanceAdvisorySwitchIsOff.java
ObjectClassRelevanceAdvisorySwitchIsOn.java
ObjectInstanceNameInUse.java
ObjectInstanceNameNotReserved.java
ObjectInstanceNotKnown.java
OwnershipAcquisitionPending.java
RTIexception.java
RTIinternalError.java
RegionDoesNotContainSpecifiedDimension.java
RegionInUseForUpdateOrSubscription.java
RegionNotCreatedByThisFederate.java
RequestForTimeConstrainedPending.java
RequestForTimeRegulationPending.java
RestoreInProgress.java
RestoreNotInProgress.java
RestoreNotRequested.java

SaveInProgress.java
SaveNotInProgress.java
SaveNotInitiated.java
SpecifiedSaveLabelDoesNotExist.java
SynchronizationPointLabelNotAnnounced.java
TimeConstrainedAlreadyEnabled.java
TimeConstrainedIsNotEnabled.java
TimeRegulationAlreadyEnabled.java
TimeRegulationIsNotEnabled.java
UnableToPerformSave.java
UnknownName.java
UnsupportedCallbackModel.java

hla/rti1516e/time:
HLAfloat64Interval.java
HLAfloat64Time.java
HLAfloat64TimeFactory.java
HLAinteger64Interval.java
HLAinteger64Time.java
HLAinteger64TimeFactory.java

Annex C

(normative)

C++ API

The files listed below shall comprise the HLA C++ API. These files are grouped in a file named IEEE1516-2010_C++_API.zip, which is part of this specification and can be downloaded from either of the following locations:

— http://standards.ieee.org/downloads/1516/1516.1-2010/IEEE1516-2010_C++_API.zip
— http://www.sisostds.org/apis.aspx/IEEE1516-2010_C++_API.zip

rti:
Enums.h
Exception.h
FederateAmbassador.h
Handle.h
LogicalTime.h
LogicalTimeFactory.h
LogicalTimeInterval.h
NullFederateAmbassador.h
RTI1516.h
RTIambassador.h
RTIambassadorFactory.h
RangeBounds.h
SpecificConfig.h
Typedefs.h
VariableLengthData.h
encoding/
time/

rti/encoding:
BasicDataElements.h
DataElement.h
DataElementFactory.h
EncodingConfig.h
EncodingExceptions.h
HLAfixedArray.h
HLAfixedRecord.h
HLAhandle.h
HLAlogicalTime.h
HLAlogicalTimeInterval.h
HLAopaqueData.h
HLAvariableArray.h
HLAvariantRecord.h

rti/time:
HLAfloat64Interval.h
HLAfloat64Time.h
HLAfloat64TimeFactory.h

HLAinteger64Interval.h
HLAinteger64Time.h
HLAinteger64TimeFactory.h

Annex D

(normative)

Web Services API

The file named hla1516e.wsdl shall comprise the HLA Web Services API. This file is part of this specification and can be downloaded from either of the following locations:

— http://standards.ieee.org/downloads/1516/1516.1-2010/hla1516e.wsdl
— http://www.sisostds.org/apis.aspx/hla1516e.wsdl

Annex E

(informative)

Rationale

E.1 Overview

This annex contains additional information that might be useful to explain various aspects of HLA and is not a complete rationale of the HLA. Nothing in this clause is required in an HLA implementation.

This annex is organized to match the organization of the main body of this specification, and the subclause numbers in this annex correspond to the subclause numbers of the main body. Note, however, that not all subclauses in the main body have corresponding subclauses in this annex. In other words, text is present in this annex only if additional comments were deemed helpful to understanding the corresponding subclauses in the main body.

E.1.4 Background

Consideration was given to providing several iterating functions; however, there were found to be serious problems documenting and describing the iterating-based semantics of these services. Additionally, there was a problem with every redaction produced. The candidate definitions of how exceptions were handled and recovered from during the "middle" of an iteration sequence never satisfied a large enough set of users to be acceptable for retention in the specification. Each of the users wanted the behavior to meet their specific application needs, which in most cases were inconsistent and impossible to meet with a common mechanism. As a result, it seemed to make the most sense not to have any iterating services and to let users produce their own aggregate functionality with the exception response semantics they needed.

E.4 Federation management

E.4.6 *Destroy Federation Execution* service

Since there can be no joined federates for the destroy service invocation to succeed, there might be a problem if the federation execution has "zombie" or otherwise uncooperative joined federates. To aid matters in this situation, a joined federate can be resigned by another joined federate via a MOM interaction.

E.4.31 *Query Federation Restore Status* service

One way to employ federation restore status is for a federation manager federate to not invoke the *Restore Complete* service until all the other joined federates have successfully completed their restore. In this way, a federation manager federate can "throttle" the restore process as it monitors the restore status.

E.5 Declaration management (DM)

E.5.1 Overview

One reason joined federates can subscribe at multiple levels in a class hierarchy is to allow for FOM growth. As a FOM expands, a federate application need not be recoded. It can continue to subscribe at the same levels in the hierarchy while other, "newer" federate applications in the federation can subscribe at the new classes.

Some services in the interface specification contain postconditions that are less precise than one would expect. The postcondition for the *Subscribe Object Class Attributes* service, for example, states only that "the RTI has been informed of the joined federate's requested subscription." It does not say that the joined federate is now subscribed to the specified class attributes, nor is a statement to this effect found in any service postcondition.

Services with such imprecise postconditions include the following:

— *Subscribe Object Class Attributes*
— *Subscribe Interaction Class*
— *Commit Region Modifications* (old Modify Region)
— *Subscribe Object Class Attributes With Regions*
— *Subscribe Interaction Class With Regions*

These services have imprecise postconditions, similar to the postcondition quoted above, because the interface specification allows RTI implementations to decide when to determine whether a message sent by one federate should be received by another federate. One possibility is to broadcast all sent messages and perform filtering at each receiving federate. A more efficient possibility is to perform filtering when the message is sent (in other words, the message is sent only to interested federates) and again at each receiving federate (in case the state of the receiver has changed during transmission).

The problem with allowing the latter alternative is that should the receiver add to its subscriptions, that addition cannot be considered to have fully taken effect until the RTI can ensure that messages sent by any federate will take the added subscriptions into account, thus the imprecise postconditions. Subscription additions take effect locally immediately, but there will be some delay before the additions take effect throughout the federation execution. An RTI implementation must ensure that these changes take effect throughout the federation execution as soon as possible, but it will not delay the return of these service invocations until it can guarantee the subscription additions are complete.

Since federates can come and go and can add subscriptions at any wall-clock time, the only way the RTI could meet a postcondition asserting that the federate is now subscribed to the specified attributes would be to synchronize the whole federation execution so that it could make that guarantee. This is one of the places in the specification where the desire was to permit more federate asynchronous behavior. This places more responsibility on the federates to ensure that they know what they are doing and make their subscriptions in sufficient wall-clock time for them to have the desired effect in their federation execution.

E.5.1.2 Definitions and constraints for object classes and class attributes

Item (o) in 5.1.2 is a very complex sentence, and it has been suggested that it may be difficult for the reader to parse and understand. This text is an attempt to explain what it means by example.

Consider Figure E.1, the object class inheritance tree.

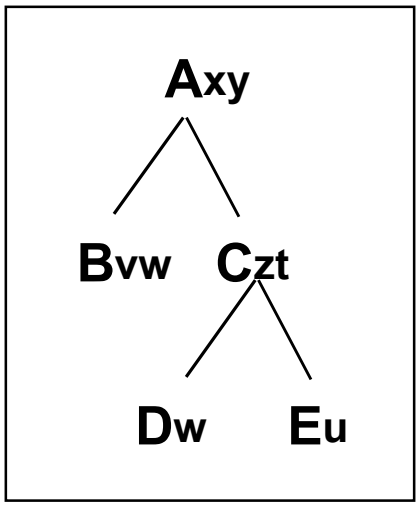

Figure E.1—Example object class inheritance tree

The following notation is used in this example:

- **Subscribe [C(z)]** denotes the invocation of the *Subscribe Object Class Attributes* service to subscribe to object class C and class attribute z.
- **Publish [E(x, u)]** denotes the invocation of the *Publish Object Class Attributes* service to publish object class E and class attributes x and u.
- **Register (E, e1)** denotes the invocation of the *Register Object Instance* service to register an object instance of class E that will be denoted e1.
- **e1(x)** denotes instance attribute x of object instance e1.

Suppose federate ***F*** invokes

- Subscribe [C(z)]
- Subscribe [A(x,y)]

Suppose federate ***H*** invokes

- Subscribe [C(x,z)]
- Subscribe [A(y)]

Suppose federate ***G*** invokes

- Publish [C(z,t)]
- Publish [E(x,y,u)]
- Register (E, e1)

Then

- Ownership: Federate ***G*** now owns e1(x), e1(y), and e1(u). Even though federate ***G*** was publishing class attributes z and t when it registered e1, federate ***G*** was not publishing z or t at the registered class of e1 (class E) when it registered e1; therefore, federate ***G*** does not own e1(z) or e1(t). e1(z) and e1(t) are not owned by any federate.

— Discovery at Federate ***F***: Federate ***F*** will not discover object instance e1. The candidate discovery class of e1 at federate ***F*** is class C. But federate ***F*** is not subscribed to any class attribute at class C that has a corresponding instance attribute that is part of e1 and that is owned. Therefore, object instance e1 is not known to federate ***F***, and it has no known class at federate ***F***. Federate ***F*** is not capable of receiving reflects for updates made to any values of the instance attributes of object instance e1. Even though federate ***F*** is subscribed to class attribute x at class A, an update made to instance attribute e1(x) will not be reflected at federate ***F*** because e1 is not known to federate ***F***. (For more information on discovery and known classes, see also 6.1.)
— Discovery at Federate ***H***: Federate ***H*** will discover object instance e1 as a class C object instance. The candidate discovery class of e1 at federate ***H*** is class C because federate ***H*** is not subscribed to e1's registered class (class E) and class C is the closest superclass of e1's registered class to which federate ***H*** is subscribed. Federate ***H*** is also subscribed to class attribute x at class C, and class attribute x has a corresponding instance attribute that is part of object instance e1 and that is owned, namely e1(x). Therefore, the known class of e1 at federate ***H*** is class C.
— Reflection: When federate ***G*** updates the value of e1(x), this update will be reflected at Federate ***H***. However, when federate ***G*** updates the value of e1(y) or e1(u), these updates will not be reflected at federate ***H*** because federate ***H*** is not subscribed to class attributes y or u at class C. Notice that even though federate ***H*** is subscribed to class attribute y, it will not receive reflects for updates made to the value of e1(y) because federate ***H*** is not subscribed to class attribute y at the known class (class C) of e1.

Recall the wording of item (o) of 5.1.2, which says, "An update to an instance attribute by the federate that owns that instance attribute shall be reflected only by other federates that are subscribed to the instance attribute's corresponding class attribute at the known class of the object instance that contains that instance attribute at the subscribing federate." In terms of this example, this wording means the following:

> An update to e1(x) by federate ***G*** shall be reflected by federate ***H*** because federate ***H*** is subscribed to class attribute x at e1's known class at federate ***H*** (namely, class C). An update to e1(y) or e1(u) by federate ***G*** shall not be reflected by federate ***H*** because federate ***H*** is not subscribed to class attributes y or u at e1's known class at H (class C).

E.6 Object management

E.6.1 Overview

For examples that further elucidate the concepts of discovery and known class, see E.5.1.2 and E.9.1.5.

If an instance attribute is in scope for a given joined federate ***F***, there exists at least one other joined federate in the federation execution that is capable of invoking the *Update Attribute Values* service for that instance attribute so that if that joined federate were to invoke the *Update Attribute Values* service for that instance attribute, all of the preconditions for the invocation of the corresponding *Reflect Attribute Values †* service at federate ***F*** are satisfied. Therefore, if an instance attribute is in scope for a given joined federate, that joined federate may receive invocations of the *Reflect Attribute Values †* service for that instance attribute. It is also possible, however, for a joined federate to receive invocations of the *Reflect Attribute Values †* service for an instance attribute that is not in scope for that joined federate. All of the conditions that define whether an instance attribute is in scope are also preconditions of the *Reflect Attribute Values †* service, with the exception of the condition that the instance attribute be owned by another joined federate. Because the conditions for determining whether an instance attribute is in scope are more restrictive than the conditions for determining whether an update to that instance attribute should be reflected, it is possible for updates to instance attributes that are not in scope to be reflected.

As an illustrative example, consider the scenario in which a joined federate that owns a particular instance attribute invokes the *Update Attribute Values* service for that instance attribute using a timestamp argument indicating a federation time far into the future. Then, that joined federate divests ownership of the instance attribute and resigns from the federation execution. No other joined federate acquires ownership of the instance attribute. When the federation time indicated in the timestamp argument of the update arrives, that instance attribute is, by definition, out of scope for all joined federates in the federation execution because it is not owned. Nevertheless, assuming all of the other preconditions for invocation of the *Reflect Attribute Values †* service for that instance attribute are satisfied at a given joined federate, the *Reflect Attribute Values †* service will be invoked at that joined federate. Hence, an instance attribute being in or out of scope for a particular joined federate indicates only whether there is some other joined federate in the federation execution that is capable of generating updates to that instance attribute so that if that joined federate were to invoke the *Update Attribute Values* service for that instance attribute, all of the preconditions for the invocation of the corresponding *Reflect Attribute Values †* service at federate ***F*** are satisfied. An instance attribute being in or out of scope for a particular joined federate makes no assertion regarding whether invocations of the *Reflect Attribute Values †* service for that instance attribute may be invoked at that joined federate.

E.6.5 *Reserve Multiple Object Instance Names* service

The multiple name reservation services provide a shortcut method for reserving multiple object instance names. Since a centralized or distributed name reservation implementation would require synchronization across the network for each name reservation transaction, this shortcut can reduce the overhead of reserving multiple object names.

E.6.14 *Delete Object Instance* service

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

E.6.19 *Request Attribute Value Update* service

A single *Request Attribute Value Update* service invocation for an object class may generate many *Provide Attribute Value Update †* service invocations at instance attribute-owning joined federates (at least one for each object instance of the class for which the joined federate owns a requested attribute). This situation will generate a nonpredictable "storm" of "interrupts" and possibly cause the timing behavior of real-time federates to become unpredictable. This might be an unwise thing to do in a real-time federation.

E.6.23 *Request Attribute Transportation Type Change* service

Actual implementation of the transportation types are RTI implementation dependent. The names may be declared in FOM/FDD.

E.8 Time management

E.8.1 Overview

At the federation level, this specification provides a variety of time management services to control the advancement of all joined federates along the HLA time axis. These services allow multiple joined federates with differing internal federation time advancement strategies to interoperate in a meaningful way. While the strategy chosen for advancing federation time is an extremely important consideration in the design of a simulation, the format of how federation time is represented is of equal importance.

It is very important for all federation members to agree upon a common set of key federation time characteristics that apply across the full federation. In addition to defining certain characteristics of the HLA time axis, it is also important to define the lookahead characteristics of both federates and federations. It is important to recognize that there are semantic differences between the way federation time is represented for the purpose of advancing federation time versus calculating lookahead. When advancing federation time, federation time can be considered to be an absolute value on a federation timeline, and thus federation time comparisons can be drawn to determine if one federation time value is greater than another. Lookahead, in contrast, represents a duration of federation time, which can be added to absolute federation time values but is not generally used for comparison purposes. For this reason, federation time is defined using two abstract datatypes (ADTs), **`LogicalTime`** and **`LogicalTimeInterval`**.

Each of the specific language APIs contain ADTs for **`LogicalTime`** and **`LogicalTimeInterval`**. These ADTs each define a set of interfaces for which implementations must be provided before a federation execution that utilizes time management services can take place. It is the responsibility of the federate/ federation developer(s) to furnish concrete implementations of **`LogicalTime`** and **`LogicalTimeInterval`** interfaces.

Additionally, each of the language APIs also provide default concrete implementations of these two ADTs based on the float 64 datatype.

E.8.1.1 Messages

While this specification states no explicit requirements on the order of delivery of RO messages, an RTI implementation may provide a more predictable order of delivery. A number of factors must be considered by an RTI when determining the order in which RO messages are received, including the following:

- Whether a series of messages sent from the same federate are delivered in the same order as the order in which they were sent
- Whether series of messages sent by different federates are interleaved
- Whether messages received as RO that were sent as RO and that were sent as TSO are interleaved

A federate that assumes a particular ordering of RO messages may not be able to freely choose among RTI implementations.

A federate may choose to send TSO messages using different transportation types, but if the delivery mechanism is unreliable, there may be circumstances in which the message must be delivered to a receiving, time-constrained federate as RO. Federation time synchronization mechanisms typically rely on reliable message delivery to efficiently advance along the HLA time axis, and messages transmitted unreliably may be delivered with a timestamp that would violate causality for a receiving federate. Under these circumstances, an RTI must deliver the TSO message as RO.

E.8.6 *Time Constrained Enabled †* service

This service description says that (if the joined federate is not time-regulating) "the RTI shall provide the joined federate with the smallest possible logical time that is greater than or equal to the logical time of the joined federate and for which all other constraints necessary to ensure TSO message delivery are satisfied." This description is overly complicated.

When a joined federate invokes the *Enable Time Constrained* service, there is no case in which the joined federate's logical time needs to be advanced. The value of the current logical time argument of the *Time Constrained Enabled †* service will always be the current logical time of the joined federate when it invoked the *Enable Time Constrained* service.

If the joined federate's current logical time is less than or equal to its GALT when it invokes the *Enable Time Constrained* service, then the *Time Constrained Enabled †* service may be invoked immediately at the joined federate. If the joined federate's current logical time is greater than its GALT when it invokes the *Enable Time Constrained* service, then the *Time Constrained Enabled †* service cannot be invoked until the joined federate's GALT advances beyond its logical time. In either case, however, when the *Time Constrained Enabled †* service is invoked at the joined federate, the value of the current logical time argument of the *Time Constrained Enabled †* service will be the value of the joined federate's current logical time when the joined federate invoked the *Enable Time Constrained* service.

E.8.12 *Flush Queue Request* service

The *Flush Queue Request* service introductory text says,

> 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 timestamp of all TSO messages delivered by the RTI in response to this invocation of the *Flush Queue Request* service

The third item in the list above is a change from the 1.3 version to the HLA specification, and the purpose of adding it is to allow federates to generate new events with "small" timestamps in response to events it received as a result of the *Flush Queue Request*. For example, consider the following scenario:

> A federate makes a *Flush Queue Request* service invocation to federation time 100.
> The federate receives a message with timestamp 95.
> The federate gets a grant to federation time T.
>
> Without the third condition above, the federate could get a grant up to federation time 100 (i.e., T=100). However, the federate may want to send a new event, e.g., *Update Attribute Values*, in response to the event with timestamp 95. This new event might have, for example, a timestamp of 96. This is a perfectly reasonable thing to do, but would cause an exception if the federate had already been advanced to federation time 100.
>
> To address this situation, the third condition will ensure that the grant will only be to federation time 95 (or less).

E.8.22 *Request Retraction †* service

Federates should be aware that, according to this service description, they may receive a *Request Retraction †* callback for a message that has not been delivered to them. In this case, federates should ignore the *Request Retraction †* callback.

E.9 Data distribution management (DDM)

E.9.1.5 Reinterpretation of selected declaration management services

See E.5.1.2 for an explanation of item (o) of 5.1.2 when regions are not being used for update of an instance attribute or used for subscription of a class attribute. The following example is intended to explain item (o) of 9.1.1 when regions are being used for update of an instance attribute or used for subscription of a class attribute.

Consider the same object class inheritance tree as pictured in Figure E.1. The following notation is used in this example in addition to that which was introduced in E.5.1.2:

— **Subscribe_w_Reg [C,R1,(x,z)]** denotes the invocation of the *Subscribe Object Class Attributes With Regions* service to subscribe to object class C and class attributes x and z and to use region R1 for subscription of class attributes x and z.
— **Register_w_Reg [E, (R1,x), (R2,y),e1]** denotes the invocation of the *Register Object Instance With Regions* service to register an object instance of class E that will be denoted e1 and to use region R1 for update of instance attribute e1(x) and region R1 for update of instance attribute e1(y).

Suppose that all class attributes in this example have the same set of available dimensions.

Suppose regions R1, R2, and R3 have been created, and each of these regions contains all of the available dimensions of the class attributes in this example.

Suppose that

— Regions R1 and R2 overlap
— Regions R1 and R3 do not overlap

Suppose federate ***H*** invokes

— Subscribe_w_Reg [C,R1,(x,z,t)]
— Subscribe_w_Reg [A,R1,(y)]

Suppose federate ***G*** invokes

— Publish [E(x,y,z,t,u)]
— Register_w_Reg [E, (R2,x),(R2,y),(R3,z),e1]

Then

— Ownership: Federate ***G*** now owns e1(x), e1(y), e1(z), e1(t), and e1(u). Region R2 is used for update of e1(x) and e1(y) by federate ***G***; region R3 is used for update of e1(z) by federate ***G***; and the default region is used for update of e1(t) and e1(u) by federate ***G***. By definition, R1, R2, and R3 all overlap with the region used for update of e1(t) and e1(u).

— Discovery at federate ***H***: Federate ***H*** will discover object instance e1 as a class C object instance. The candidate discovery class of e1 at federate ***H*** is class C because federate ***H*** is not subscribed to e1's registered class (class E) and class C is the closest superclass of e1's registered class to which federate ***H*** is subscribed. Federate ***H*** is also subscribed to class attribute x at class C with region R1, and class attribute x has a corresponding instance attribute that is part of object instance e1 and that is owned (namely e1(x)). Region R2 is used for update of e1(x) by its owning federate (federate ***G***), and regions R1 and R2 overlap. Therefore, the known class of e1 at federate ***H*** is class C.

— Reflection: When federate ***G*** updates the value of e1(x) or e1(t), these updates shall be reflected at federate ***H***. Updates to e1(x) and e1(t) will be reflected at federate ***H*** because federate ***H*** is subscribed to class attributes x and t at e1's known class (class C), and the region that is used for subscription of x and t by federate ***H*** (region R1) overlaps the region that is used for updates of e1(x) and e1(t) by owning federate ***G*** (namely, region R2 in the case of e1(x) and the default region of routing space R in the case of e1(t)). However, when federate ***G*** updates the value of e1(y), e1(z), or e1(u), these updates will not be reflected at federate ***H***. Updates to e1(y) and e1(u) will not be reflected at federate ***H*** because federate ***H*** is not subscribed to class attribute y at object instance e1's known class (class C) and class attribute u is not even available for subscription at object instance e1's known class (class C). Federate ***H*** is subscribed with region to class attribute z at e1's known class, but updates to e1(z) will not be reflected at federate ***H*** because federate ***H*** is using region R1 for subscription of class attribute z, federate ***G*** is using region R3 for update of instance attribute e1(z), and these two regions (R1 and R3) do not overlap.

E.9.13 *Request Attribute Value Update With Regions* service

Unlike the corresponding object management service (*Request Attribute Value Update*), this service does not allow an object instance to be used as an argument; only an object class can be used. If an update for attributes of a specific object instance is needed, the object management service should be used. As a request that takes an object instance is already quite specific, additional DDM filtering based on regions is not needed for such cases.

A single *Request Attribute Value Update With Regions* service invocation for an object class may generate many *Provide Attribute Value Update †* service invocations at instance attribute-owning joined federates (at least one for each object instance of the class for which the joined federate owns a requested attribute). This situation will generate a nonpredictable "storm" of "interrupts" and possibly cause the timing behavior of real-time federates to become unpredictable. This might be an unwise thing to do in a real-time federation.

E.11 Management object model (MOM)

The defined MOM functionality is necessary for successful federation execution operation and can be provided only by the RTI. The MOM definition needs to be standardized by defining it in this specification.

Remotely changing joined federate states (e.g., publication, transportation types, time management) as provided by the MOM can be error prone. However, these sorts of actions are needed in federations to recover from catastrophic error conditions. They should be used with extreme caution.

Additionally, there are ways a federate can ascertain that MOM has done something "on its behalf." The "target" federate can subscribe to **ModifyAttributeState** and all **Manager.Federate.Service** interactions and inspect them to determine if other federates are attempting to change its owned attribute state or invoke HLA services on its behalf. The only issue is that the RTI may react to the MOM interaction before the affected federate receives the particular notifying interaction.

E.11.6 MOM OMT tables

In Table 4 (in 11.6), the HLAObjectRoot object is present as a class in which to house the **HLAprivilegeToDeleteObject** object class attribute. Additionally, the HLAObjectRoot class provides a coinvent soubriquet with which to reference the entire object class tree.

In Table 5 (in 11.6), the HLAinteractionRoot class provides a coinvent soubriquet with which to reference the entire interaction class tree.

The value of the HLAfederateType class attribute of the object class HLAmanager.HLAfederate is actually a joined federate type, since the federate “indicator” argument to the *Join Federation Execution* service is a federate type and not a federate name.

Annex F

(normative)

FOM Document Data (FDD) XML Schema declaration

The XML specification defines two levels of correctness for XML documents: well-formed and valid. A well-formed XML document is one that follows the syntax constraints of XML; for example, all element opening and closing tags are appropriately nested. A valid XML file is one that is well formed and complies with an XML Schema.

The XML Schema declaration for the FDD shall be defined in the file named IEEE1516-FDD-2010.xsd. This file is part of this specification and can be downloaded from either of the following locations:

— http://standards.ieee.org/downloads/1516/1516.1-2010/IEEE1516-FDD-2010.xsd
— http://www.sisostds.org/schemas.aspx/IEEE1516-FDD-2010.xsd

An FDD shall be considered valid if and only if it complies with the FDD XML Schema. All compliant RTI implementations shall be able to read valid FDDs.

The FDD Schema represents the minimum set of FOM information required to initialize a federation. The Schema allows additional FOM information to be present in the FDD. Furthermore, the schema allows the FDD to contain additional non-FOM elements and attributes, e.g., RTI specific information.

Annex G

(normative)

MOM and Initialization Module (MIM)

The MIM shall be defined in the file named HLAstandardMIM.xml. This file is part of this specification and can be downloaded from either of the following locations:

— http://standards.ieee.org/downloads/1516/1516.1-2010/HLAstandardMIM.xml
— http://www.sisostds.org/schemas.aspx/HLAstandardMIM.xml

If there are any differences between the MOM tables in the XML defined in that file and the tables defined in Clause 11, the clause (i.e., Clause 11) shall take precedence.

Annex H

(informative)

Bibliography

[B1] Austern, Matthew H., *Generic Programming and the STL, Using and Extending the C++ Standard Template Library*. Reading, MA: Addison-Wesley, 1999.

[B2] Harel, David, "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming* (Netherlands), vol. 8, no. 3, pp. 231–274, June 1987.

[B3] ISO/IEC 14882:1998, Programming Languages—C++.

[B4] Meyers, Scott, *Effective C++: 50 Specific Ways to Improve Your Programs and Designs*, 2nd ed. Reading, MA: Addison-Wesley, 1997.

[B5] SISO-STD-004.1-2004, Dynamic Link Compatible HLA API Standard for the HLA Interface Specification (IEEE 1516.1 Version), dated 20 April 2006.

[B6] Stroustrup, Bjorne, *The C++ Programming Language*, 3rd ed. Reading, MA: Addison-Wesley, 1997.

[B7] IEEE 1516.3™-2003, IEEE Recommended Practice For High Level Architecture (HLA) Federation Development and Execution Process (FEDEP).[10,11]

[B8] IEEE 1516.4™-2007, IEEE Recommended Practice for Verification, Validation, and Accreditation of a Federation—An Overlay to the High Level Architecture Federation Development and Execution Process.

[10]IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, Piscataway, NJ 08854, USA (http://standards.ieee.org/).

[11]The IEEE standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

Annex I

(informative)

Figure table

Table 29—Figure numbers