

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

Sponsor

Simulation Interoperability Standards Committee
of the
IEEE Computer Society

Approved 21 September 2000

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, high level architecture (HLA), instance attribute, instance attribute ownership, interaction class, object class, runtime infrastructure (RTI), simulation object model, time-constrained, time-regulating

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

Copyright © 2001 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 9 March 2001. Printed in the United States of America.

Print: ISBN 0-7381-2621-7 SH94883
PDF: ISBN 0-7381-2622-5 SS94883

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

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 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. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

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 the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

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.

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

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

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

IEEE is the sole entity that may authorize the use of certification marks, trademarks, or other designations to indicate compliance with the materials set forth herein.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; (978) 750-8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

(This introduction is not part of the IEEE Std 1516.1-2000, 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 M&S 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.

Participants

This standard was prepared by the HLA Working Group, and sponsored by the Simulation Interoperability Standards Committee (SISC) of the IEEE Computer Society. When the working group approved this standard, it had the following membership:

Susan F. Symington, *Chair*
Katherine L. Morse, *Vice Chair*
Mikel Petty, *Secretary*

Stephen Bachinsky
Joe Batman
Brian Beebe
Keith Briggs
Jim Calvin
Richard M. Fujimoto
Deborah Fullford
Allison Griffin

James H. Hammond
James Ivers
Andreas Kemkes
John F. Kramer
Frederick S. Kuhl
Gary M. Lightner
Reed Little
Rodney Long
Margaret Loper
Robert R. Lutz

Jeffery M. Nielsen
Marnie R. Salisbury
Randy Saunders
Roy Scrudder
David W. Seidel
Robert C. Turrell
Richard Weatherly
Annette Wilson

The working group acknowledges the following individuals who also contributed to the preparation of this standard:

Phillippe Annic
Tobin Bergen-Hill
Dominique Canazzi

Judith S. Dahmann
Peter Hoare
Mikael Karlsson

Michael Mazurek
J. Russell Noseworthy
Graham Shanks

The following members of the balloting committee voted on this standard:

Dale E. Anglin	Ken Hunt	Marnie R. Salisbury
Stephen Bachinsky	Kyle Isakson	Randy Saunders
Michael R. Bachmann	James Ivers	David W. Seidel
David L. Barton	Andreas Kemkes	Richard J. Severinghaus
Robert P. Bennett	John F. Kramer	Joseph F. Seward
Christina L. Bouwens	Frederick S. Kuhl	Sean Sharp
Keith Briggs	Tom Lake	Steven M. Sheasby
Richard Briggs	Gary M. Lightner	John W. Sheppard
Danny Cohen	Paul R. Little	Allen H. Skees
Glenn Conrad	Reed Little	Col. Mark Smith
Mark Crooks	Margaret Loper	Susan D. Solick
Dannie Cutts	Robert R. Lutz	Sandra Swearingen
Judith S. Dahmann	Jayne Lyons	Stephen J. Swenson
Steven Drake	Theodore F. Marz	Susan F. Symington
Laura E. Feinerman	Mark McAuliffe	Donald Theune
Jeff E. Fischer	David R. McKeeby	William V. Tucker
Richard M. Fujimoto	Katherine L. Morse	Grant R. Tudor
Deborah Fullford	Thomas H. Mullins	John A. Tufarolo
Brian F. Goldiez	Jeffrey M. Nielsen	Robert C. Turrell
Victor M. Gonzalez	Michael J. O'Connor	Andrew J. Ventre
James H. Hammond	John Peng	William F. Waite
Jack G. Harrington	Paul Perkinson	Charles E. Walters
Robert V. Head, Jr.	Hung Q. Phan	Richard Weatherly
Callie M. Hill	David R. Pratt	Chris C. Wertman
Ronald C. Hofer	Paul M. Reeves	Annette Wilson
James W. Hollenbach	David Roberts	Douglas D. Wood
Margaret M. Horst		Philomena M. Zimmerman

When the IEEE-SA Standards Board approved this standard on 21 September 2000, it had the following membership:

Donald N. Heirman, *Chair*

James T. Carlo, *Vice Chair*

Judith Gorman, *Secretary*

Satish K. Aggarwal	James H. Gurney	James W. Moore
Mark D. Bowman	Richard J. Holleman	Robert F. Munzner
Gary R. Engmann	Lowell G. Johnson	Ronald C. Petersen
Harold E. Epstein	Robert J. Kennelly	Gerald H. Peterson
H. Landis Floyd	Joseph L. Koepfinger*	John B. Posey
Jay Forster*	Peter H. Lips	Gary S. Robinson
Howard M. Frazier	L. Bruce McClung	Akio Tojo
Ruben D. Garzon	Daleep C. Mohla	Donald W. Zipse

*Member Emeritus

Also included is the following nonvoting IEEE-SA Standards Board liaison:

Alan Cookson, *NIST Representative*

Donald R. Volzka, *TAB Representative*

Jennifer McClain Longman
IEEE Standards Project Editor

Contents

1.	Overview.....	1
1.1	Scope.....	1
1.2	Purpose.....	1
1.3	Introduction.....	1
1.4	Background.....	2
2.	References.....	5
3.	Definitions, abbreviations, and acronyms.....	6
3.1	Definitions	6
3.2	Abbreviations and acronyms	17
4.	Federation management.....	18
4.1	Overview.....	18
4.2	Create Federation Execution.....	27
4.3	Destroy Federation Execution	27
4.4	Join Federation Execution	28
4.5	Resign Federation Execution	29
4.6	Register Federation Synchronization Point	31
4.7	Confirm Synchronization Point Registration †.....	32
4.8	Announce Synchronization Point †	33
4.9	Synchronization Point Achieved	34
4.10	Federation Synchronized †	34
4.11	Request Federation Save.....	35
4.12	Initiate Federate Save †.....	37
4.13	Federate Save Begun	38
4.14	Federate Save Complete	38
4.15	Federation Saved †.....	39
4.16	Query Federation Save Status.....	40
4.17	Federation Save Status Response †	41
4.18	Request Federation Restore	42
4.19	Confirm Federation Restoration Request †	43
4.20	Federation Restore Begun †.....	44
4.21	Initiate Federate Restore †	44
4.22	Federate Restore Complete.....	45
4.23	Federation Restored †	46
4.24	Query Federation Restore Status	47
4.25	Federation Restore Status Response †	48
5.	Declaration management	49
5.1	Overview.....	49
5.2	Publish Object Class Attributes	56
5.3	Unpublish Object Class Attributes	58
5.4	Publish Interaction Class	59
5.5	Unpublish Interaction Class.....	60
5.6	Subscribe Object Class Attributes	61
5.7	Unsubscribe Object Class Attributes	63
5.8	Subscribe Interaction Class.....	64
5.9	Unsubscribe Interaction Class	65

5.10	Start Registration For Object Class †	66
5.11	Stop Registration For Object Class †	67
5.12	Turn Interactions On †	68
5.13	Turn Interactions Off †	69
6.	Object management	70
6.1	Overview	70
6.2	Reserve Object Instance Name	74
6.3	Object Instance Name Reserved †	75
6.4	Register Object Instance	76
6.5	Discover Object Instance †	77
6.6	Update Attribute Values	78
6.7	Reflect Attribute Values †	80
6.8	Send Interaction	81
6.9	Receive Interaction †	83
6.10	Delete Object Instance	84
6.11	Remove object instance †	86
6.12	Local Delete Object Instance	87
6.13	Change Attribute Transportation Type	88
6.14	Change Interaction Transportation Type	90
6.15	Attributes In Scope †	91
6.16	Attributes Out Of Scope †	92
6.17	Request Attribute Value Update	93
6.18	Provide Attribute Value Update †	94
6.19	Turn Updates On For Object Instance †	95
6.20	Turn Updates Off For Object Instance †	96
7.	Ownership management	97
7.1	Overview	97
7.2	Unconditional Attribute Ownership Divestiture	103
7.3	Negotiated Attribute Ownership Divestiture	104
7.4	Request Attribute Ownership Assumption †	105
7.5	Request Divestiture Confirmation †	106
7.6	Confirm Divestiture	107
7.7	Attribute Ownership Acquisition Notification †	108
7.8	Attribute Ownership Acquisition	109
7.9	Attribute Ownership Acquisition If Available	110
7.10	Attribute Ownership Unavailable †	112
7.11	Request Attribute Ownership Release †	113
7.12	Attribute Ownership Divestiture If Wanted	114
7.13	Cancel Negotiated Attribute Ownership Divestiture	115
7.14	Cancel Attribute Ownership Acquisition	116
7.15	Confirm Attribute Ownership Acquisition Cancellation †	117
7.16	Query Attribute Ownership	118
7.17	Inform Attribute Ownership †	119
7.18	Is Attribute Owned By Federate	119
8.	Time management	121
8.1	Overview	121
8.2	Enable time regulation	130
8.3	Time Regulation Enabled †	132
8.4	Disable Time Regulation	133

8.5	Enable Time Constrained.....	133
8.6	Time Constrained Enabled †.....	134
8.7	Disable Time Constrained	135
8.8	Time Advance Request.....	136
8.9	Time Advance Request Available	138
8.10	Next Message Request.....	139
8.11	Next Message Request Available	141
8.12	Flush Queue Request	143
8.13	Time Advance Grant †.....	145
8.14	Enable Asynchronous Delivery	146
8.15	Disable Asynchronous Delivery	146
8.16	Query GALT.....	147
8.17	Query Logical Time.....	148
8.18	Query LITS.....	149
8.19	Modify Lookahead.....	149
8.20	Query Lookahead.....	150
8.21	Retract.....	151
8.22	Request Retraction †.....	153
8.23	Change Attribute Order Type	153
8.24	Change Interaction Order Type	155
9.	Data distribution management.....	156
9.1	Overview.....	156
9.2	Create Region	164
9.3	Commit Region Modifications	165
9.4	Delete Region	166
9.5	Register Object Instance With Regions.....	167
9.6	Associate Regions For Updates	169
9.7	Unassociate Regions For Updates	170
9.8	Subscribe Object Class Attributes With Regions	171
9.9	Unsubscribe Object Class Attributes With Regions.....	173
9.10	Subscribe Interaction Class With Regions.....	174
9.11	Unsubscribe Interaction Class With Regions	176
9.12	Send Interaction With Regions.....	177
9.13	Request Attribute Value Update With Regions.....	179
10.	Support services.....	181
10.1	Overview.....	181
10.2	Get Object Class Handle.....	182
10.3	Get Object Class Name.....	183
10.4	Get Attribute Handle.....	183
10.5	Get Attribute Name.....	184
10.6	Get Interaction Class Handle	185
10.7	Get Interaction Class Name	186
10.8	Get Parameter Handle.....	186
10.9	Get Parameter Name	187
10.10	Get Object Instance Handle	188
10.11	Get Object Instance Name	189
10.12	Get Dimension Handle.....	189
10.13	Get Dimension Name.....	190
10.14	Get Dimension Upper Bound	191
10.15	Get Available Dimensions For Class Attribute	191

10.16	Get Known Object Class Handle	192
10.17	Get Available Dimensions For Interaction Class	193
10.18	Get Transportation Type	194
10.19	Get Transportation Name.....	194
10.20	Get Order Type	195
10.21	Get order name.....	196
10.22	Enable Object Class Relevance Advisory Switch	196
10.23	Disable Object Class Relevance Advisory Switch	197
10.24	Enable Attribute Relevance Advisory Switch	198
10.25	Disable Attribute Relevance Advisory Switch	199
10.26	Enable Attribute Scope Advisory Switch	199
10.27	Disable Attribute Scope Advisory Switch	200
10.28	Enable Interaction Relevance Advisory Switch	201
10.29	Disable Interaction Relevance Advisory Switch	202
10.30	Get Dimension Handle Set	202
10.31	Get Range Bounds	203
10.32	Set Range Bounds.....	204
10.33	Normalize Federate Handle	205
10.34	Normalize Service Group	206
10.35	Initialize RTI.....	206
10.36	Finalize RTI	207
10.37	Evoke Callback.....	208
10.38	Evoke Multiple Callbacks.....	208
10.39	Enable Callbacks.....	209
10.40	Disable Callbacks	210
11.	Management object model (MOM)	211
11.1	Overview.....	211
11.2	MOM object classes.....	211
11.3	MOM interaction classes	213
11.4	MOM-related characteristics of the RTI.....	214
11.5	Service-reporting	216
11.6	MOM OMT tables	216
12.	Programming language mappings	248
12.1	Overview.....	248
12.2	Designators	248
12.3	Ada 95.....	249
12.4	Java	260
12.5	C++	272
Annex A	(normative) Ada 95 application programmer's interface	285
Annex B	(normative) Java application programmer's interface	331
Annex C	(normative) C++ application programmer's interface	391
Annex D	(informative) Rationale.....	449
Annex E	(informative) Bibliography	461
Index		462

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. 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 modeling and simulation (M&S) high level architecture comprises three main components: the HLA rules, the HLA federate interface specification, and the HLA object model template (OMT). This document shall provide a complete description of the essential elements of the second component of the HLA, the interface specification. The other two components of the HLA formal definition are described in the documents listed in Clause 2.

The high level architecture 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 as follows:

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

The seven service groups describe the interface between the federates and the RTI, 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).

Any software acting as an HLA RTI shall implement all of the federate initiated services in accordance with this specification. Likewise, any software acting as an HLA federate shall implement all of the RTI initiated federate services in accordance with this specification.

1.4 Background

1.4.1 HLA federation object model 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 federation object model (FOM) are described in IEEE Std 1516.2-2000. 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.

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 is outside the purview of the HLA federation object model.

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 federation object model 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 an RTI, and using various execution details.

1.4.2 Relationship of HLA and object-oriented concepts

Although the HLA OMT is the standardized documentation structure for HLA object models, FOMs and SOMs do not completely correspond to common definitions of object models in object-oriented (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, focusing specifically on requirements and capabilities for federate information exchange. For SOMs, 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, 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 (object classes, attributes, interaction classes, parameters, dimensions, regions, federates, object instances, etc.) referenced in this document that may have the following different “views:”

- Name: human readable or for communication between federates.
- Handle: capable of being manipulated by a computer or for communication between a federate and the RTI. 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 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 shall be needed for the implementation of a running RTI and federation executions:

- FOM Document Data (FDD): FOM information (class, attribute, parameter names, etc.) used by the RTI at runtime. Each federation execution needs an FDD. In the abstract, creation of a federation execution is simply the binding of a federation execution name to an FDD.

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, a RID is supplied when an RTI is initialized.

For all joined federate-initiated services in this specification, except 4.2, *Create Federation Execution*, 4.3, *Destroy Federation Execution*, and 4.4, *Join Federation Execution*, an implied supplied argument that is a joined federate's connection to 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, and 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.

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 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.
- Reference state charts: reference to state chart(s) that are germane to the service.

After the clauses describing each of the service groups is a clause describing the management object model (MOM) and then a clause that ties the service narratives with the APIs.

Annex A through Annex C contain HLA application programmer's interfaces (APIs) for the following languages:

- Ada 95
- Java
- C++

Annex D and Annex E contain a bibliography and rationale, respectively.

2. References

This standard shall be used in conjunction with the following publications. When the following specifications are superseded by an approved revision, the revision shall apply.

IEEE Std 1516-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Framework and Rules.¹

IEEE Std 1516.2-2000, 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.

¹IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (<http://standards.ieee.org/>).

3. Definitions, abbreviations, and acronyms

3.1 Definitions

For the purposes of this standard, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards Terms* [B6]² should be referenced for terms not defined in this clause.

3.1.1 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.

3.1.2 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 this 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.

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

3.1.4 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**.

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

3.1.6 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 in IEEE Std 1516.1-2000.

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

3.1.8 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. The term candidate discovery class pertains to object instances only.

²The numbers in brackets correspond to those of the bibliography in Annex E.

3.1.9 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. The term candidate received class pertains to interactions only.

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

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

3.1.12 class hierarchy: A specification of a class–subclass or “is–a” relationship between classes in a given domain.

3.1.13 coadunate: To attach an object instance handle (and possibly an object instance name) to an object instance.

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

3.1.15 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.

3.1.16 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).

NOTE—See IEEE Std 1516.2-2000.

3.1.17 constrained set of pairs: A group of pairs in which no two pairs have the same first component (this 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.

3.1.18 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.

3.1.19 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.

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

3.1.21 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.

3.1.22 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.

3.1.23 default range: A range lower bound and a range upper bound, defined in the Federation Object Model Document Data (FDD) and specified in terms of [0, the dimension's upper bound), for a dimension.

3.1.24 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 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 in IEEE Std 1516.1-2000.

3.1.25 designator: Some arguments (mostly identifiers) to services may have different views (implementations), depending on a particular programming language, application programmer's interface (API). For clarity, service descriptions refer to a generic view of the arguments, known as a *designator*.

3.1.26 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 upperbound as specified in the Federation Object Model 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 of IEEE Std 1516.1-2000.

3.1.27 discover: To receive an invocation of the *Discover Object Instance* ^{†3} service for a particular object instance.

NOTE—See 6.5 of IEEE Std 1516.1-2000.

3.1.28 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 in IEEE Std 1516.1-2000.

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

3.1.30 federate: An application that may be or is currently coupled with other software applications under a Federation Object Model Document Data (FDD) and a runtime infrastructure (RTI). *See also:* **federate application** and **joined federate**.

3.1.31 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**.

3.1.32 federation: A named set of federate applications and a common Federation Object Model that are used as a whole to achieve some specific objective.

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

³All RTI initiated services are denoted with a † (printer's dagger) after the service name.

3.1.34 Federation Object Model (FOM): A specification defining the information exchanged at runtime to achieve a given set of federation objectives. This includes object classes, object class attributes, interaction classes, interaction parameters, and other relevant information.

3.1.35 Federation Object Model (FOM) Document Data (FDD): The data and information in an FOM document that is used by the *Create Federation Execution* service to initialize a newly created federation execution

NOTE—See IEEE Std 1516.2-2000.

3.1.36 handle: An identifier originated/created by the runtime infrastructure (RTI) that is federation execution-wide unique and unpredictable.

3.1.37 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.

3.1.38 in scope: Of or pertaining to an instance attribute of an object instance for which

- a) The object instance is known to the joined federate
- b) The instance attribute is owned by another joined federate, and
- c) Either the instance attribute's corresponding class attribute is a
 - 1) Subscribed attribute of the known class of the object instance, or
 - 2) 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 of IEEE Std 1516.1-2000.

3.1.39 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).

3.1.40 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).

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

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

3.1.43 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.

3.1.44 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.

3.1.45 joined federate: A member of a federation execution, actualized by a federate application invoking the *Join Federation Execution* service as prescribed in IEEE Std 1516.1-2000. *See also:* **federate application.**

3.1.46 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.

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

3.1.48 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 time stamps can be sent in time stamp order (TSO) messages (relative to their logical time) to ensure that federates receiving those messages receive them in TSO.

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

3.1.50 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), and
- c) Control of the functioning of the RTI, the federation execution, and the individual joined federates

3.1.51 message: A change of object instance attribute value, 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 time stamp.

3.1.52 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. 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.

3.1.53 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.

3.1.54 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.

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

3.1.56 order type: A runtime infrastructure (RTI) provided 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.

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

- a) The object instance is known to the joined federate
- b) The instance attribute is owned by another joined federate, and
- c) Either the instance attribute's corresponding class attribute is a
 - 1) Subscribed attribute of the known class of the object instance, or
 - 2) 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 of IEEE Std 1516.1-2000.

3.1.58 overlap:

- a) Pertaining to region sets: Two region sets overlap if there is a region in each set, such that the two regions overlap.
- b) Pertaining to regions: If two regions have at least one dimension in common, they overlap if all ranges of dimensions that are contained in both regions overlap pairwise. If two regions do not have any dimensions in common, they do not overlap.
- c) Pertaining to ranges: Two ranges $A = [a_{\text{lower}}, a_{\text{upper}})$ and $B = [b_{\text{lower}}, b_{\text{upper}})$ overlap, if and only if either $a_{\text{lower}} = b_{\text{lower}}$ or $(a_{\text{lower}} < b_{\text{lower}} < a_{\text{upper}})$.

NOTE—See 9.1 of IEEE Std 1516.1-2000.

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

3.1.60 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 Document Data (FDD).

3.1.61 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.

3.1.62 parameter: A named characteristic of an interaction.

3.1.63 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, time stamp (if present), and set of sent region designators (if present). A passel is a message.

3.1.64 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, but 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.

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

NOTE—See 5.1.3 of IEEE Std 1516.1-2000.

3.1.66 published:

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

NOTE—See 5.1.2 of IEEE Std 1516.1-2000.

- b) Pertaining to an interaction class that, from the perspective of a given joined federate, was 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 of IEEE Std 1516.1-2000.

3.1.67 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 **HLA privilege ToDeleteObject** attribute for that object class.

NOTE—See 5.1.2 of IEEE Std 1516.1-2000.

3.1.68 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 of IEEE Std 1516.1-2000.

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

NOTE—See 9.1 of IEEE Std 1516.1-2000.

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

NOTE—See IEEE 9.1 of Std 1516.1-2000.

3.1.71 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 time stamp value will be provided with the message if one was specified when the message was sent, but that time stamp has no bearing on message receipt order.

3.1.72 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 of IEEE Std 1516.1-2000.

3.1.73 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 of IEEE Std 1516.1-2000.

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

NOTE—See 6.7 of IEEE Std 1516.1-2000.

3.1.75 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.

3.1.76 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 of IEEE Std 1516.1-2000.

3.1.77 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 of IEEE Std 1516.1-2000.

3.1.78 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 of IEEE Std 1516.1-2000.

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

NOTE—See 6.4 of IEEE Std 1516.1-2000.

3.1.80 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.

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

3.1.82 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.

3.1.83 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.

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

3.1.85 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 of IEEE Std 1516.1-2000.

3.1.86 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.

3.1.87 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 of IEEE Std 1516.1-2000.

3.1.88 set: A group of elements in which each element occurs at most once (this 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.

3.1.89 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 standard format in which SOMs are expressed facilitates determination of the suitability of federates for participation in a federation.

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

NOTE—See 9.1.2 of IEEE Std 1516.1-2000.

3.1.91 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.

3.1.92 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.

3.1.93 subscribed:

- a) Pertaining to an object class for which, from the perspective of a given joined federate, there are subscribed attributes of that class or subscribed attributes of that class with regions, for some region. *See also:* **subscribed attributes of an object class** and **subscribed attributes of an object class with regions**.
- b) Pertaining to an interaction class that is a subscribed interaction class or a subscribed interaction class with regions, for some region. *See also:* **subscribed interaction class** and **subscribed interaction class with regions**.

3.1.94 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 of IEEE Std 1516.1-2000.

3.1.95 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 of IEEE Std 1516.1-2000.

3.1.96 subscribed interaction class: Pertaining to 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 of IEEE Std 1516.1-2000.

3.1.97 subscribed interaction class with regions: Pertaining to 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 of IEEE Std 1516.1-2000.

3.1.98 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 and used for subscription of an interaction class.*

3.1.99 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 only inherit from a single class at the next highest level of the class hierarchy).

3.1.100 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, if they are successful, thereby synchronize their respective processing at that point.

3.1.101 time advancing state: A joined federate may advance its logical time only by requesting a time advancement from the runtime infrastructure (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. During the interval between a request to advance its logical time and the corresponding grant, the joined federate is in the time advancing state.

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

NOTE—See 8.1 of IEEE Std 1516.1-2000.

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

NOTE—See 8.1 of IEEE Std 1516.1-2000.

3.1.104 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.

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

3.1.106 time stamp order (TSO): An ordering of messages provided by a runtime infrastructure (RTI) for joined federates making use of time management services and messages containing time stamps. Messages having different time stamps are said to be delivered in TSO if for any two messages M1 and M2 (time stamped 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 time stamp will be delivered in an arbitrary order (i.e., no tie-breaking mechanism is provided by an RTI).

3.1.107 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.

3.1.108 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 of IEEE Std 1516.1-2000.

3.1.109 update: Invoke the *Update Attribute Values* service for one or more instance attributes.

NOTE—See 6.6 of IEEE Std 1516.1-2000.

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

3.1.111 used for sending:

- a) Pertaining to a region that, along with the specified interaction class designator, is being used as an argument in the *Send Interaction With Regions* service.
- b) Pertaining to the default region when the specified interaction class designator is being used as an argument in the *Send Interaction* service.

NOTE—See 9.1 of IEEE Std 1516.1-2000.

3.1.112 used for subscription of a class attribute:

- a) Pertaining to a region, an object class, and a class attribute for which 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 of IEEE Std 1516.1-2000.

- 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 of IEEE Std 1516.1-2000.

3.1.113 used for subscription of an interaction class:

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

NOTE—See 9.1 of IEEE Std 1516.1-2000.

- 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 of IEEE Std 1516.1-2000.

3.1.114 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; and 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 of IEEE Std 1516.1-2000.

- 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 of IEEE Std 1516.1-2000.

3.2 Abbreviations and acronyms

The following abbreviations and acronyms pertain to this standard.

API	application programmer's interface
BNF	Backus Naur Form
DDM	Data Distribution Management
DIF	data interchange format
DM	Declaration Management
DTD	Document Type Declaration
FDD	FOM Document Data
FOM	Federation Object Model
FQR	<i>Flush Queue Request</i>
GALT	Greatest Available Logical Time
HLA	High Level Architecture
LITS	Least Incoming Time Stamp
MOM	Management Object Model
M&S	modeling and simulation
NA	not applicable
NMR	<i>Next Message Request</i>
NMRA	<i>Next Message Request Available</i>
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
SISC	Simulation Interoperability Standards Committee
SOM	Simulation Object Model
TAR	<i>Time Advance Request</i>
TARA	<i>Time Advance Request Available</i>
TRADT	time representation abstract datatype
TSO	time stamp order
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.

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.⁴ 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.

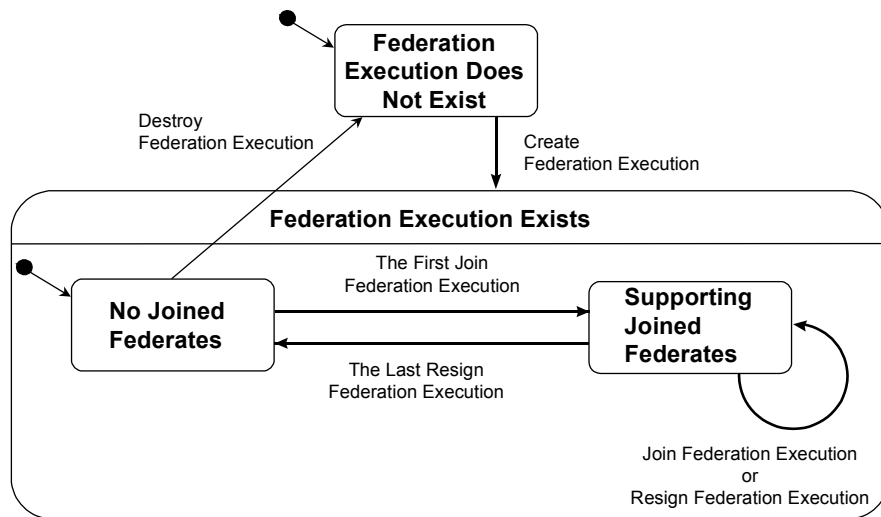


Figure 1—Basic states of the federation execution

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.

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 statechart notation pioneered by

⁴Note that this state is also reachable after all federates have resigned from a running execution; in this case, the federation also reverts to the clean, or freshly initialized state.

David Harel [B5]. This specification contains two independent sets of statecharts. The first set, beginning in Figure 3 and including all other statecharts except Figure 5 through Figure 8, describes the state of a given federate, from the perspective of that federate, in varying levels of detail. The second set, Figure 5 through Figure 8, 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 statecharts. It is a stand-alone overview of the basic states of a federation execution.

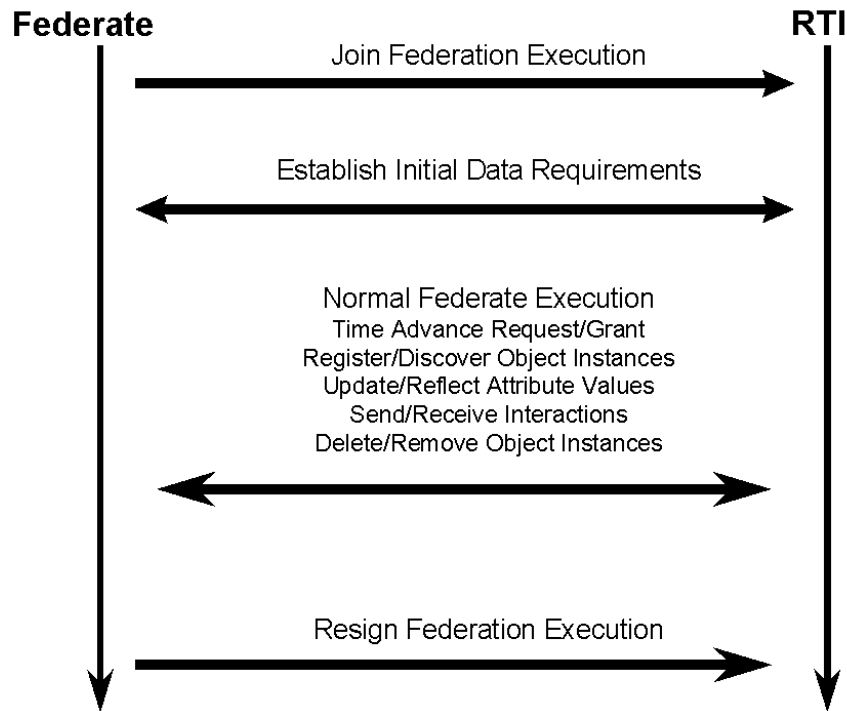


Figure 2—Overall view of federate-to-RTI relationship

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

4.1.1 Federate lifetime

As shown in Figure 3, with the successful completion of the *Join Federation Execution* service, a federate will be in the *Joined Federate* state, where it will remain until it resigns from the federation execution. As indicated by the dashed line in the *Joined Federate* state, 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* state machine, Figure 16, which appears later in this document. 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 only has a guard (with no associated service invocation), the joined federate will immediately make the associated transition as soon as the guard is true.

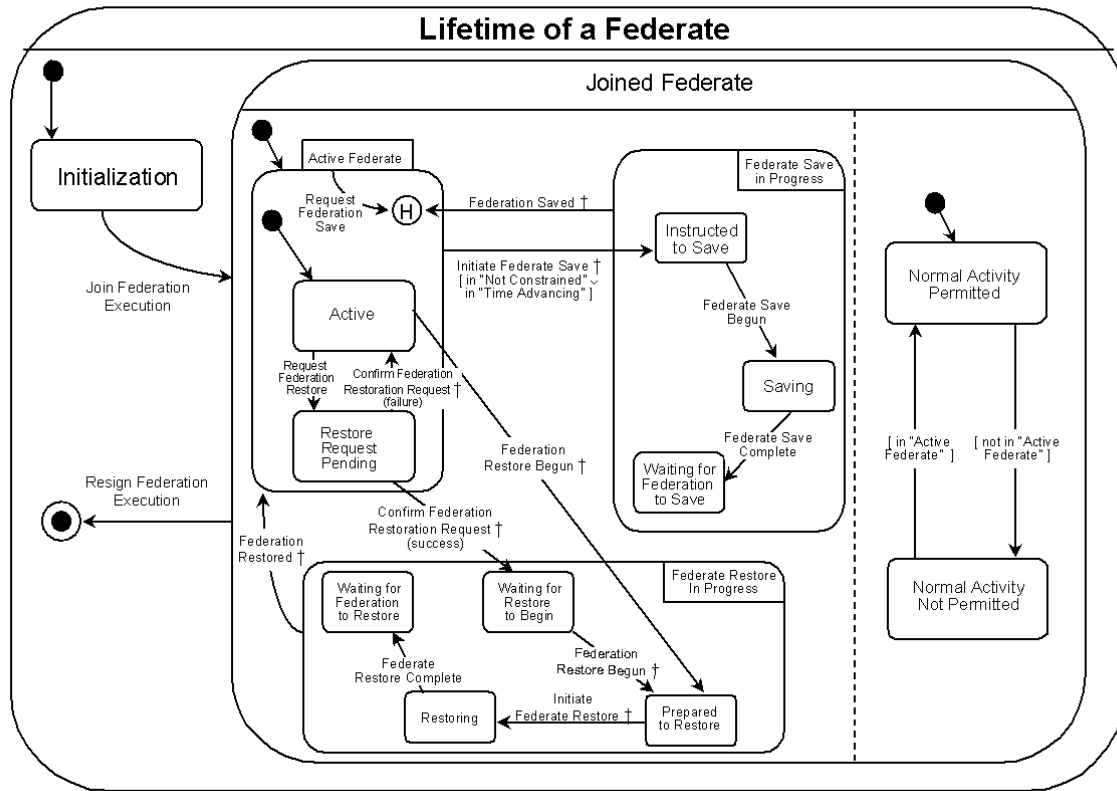


Figure 3—Lifetime of a federate

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* †⁵ 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 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 to the Normal Activity Not Permitted state). That is, 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 either 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).

The interdependency between the state machine on the left and the Temporal State state machine, Figure 16, depicted later in this document, 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 or the Time Advancing state. The Not Constrained and Time Advancing states are depicted in Figure 16. The fact that these two time management-related states are mentioned in the guard on the transition from the Active 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

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

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* $\dagger$ service.

There is a statechart notation that needs additional explanation, that of the shallow history state, depicted by a state labeled **H**. This 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 is the first occurrence in. For example, in Figure 3 there is a shallow history state within the Active Federate state. The transition for Request Federation Save originates from the outer edge of the Active Federate state, meaning that 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, meaning that on completing the transition, the federate will return to whatever state within Active Federate it was previously in (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 super-state and will result in the machine returning to the same state it was in 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. The Normal Activity Permitted state, simple as it may appear in the Joined Federate statechart, actually contains all of the other states that appear in the first set of statecharts that appear subsequently in this document. Together, these statecharts formally describe the state of a joined federate from that joined federate's perspective. These statecharts are complete in the sense that all transitions shown represent legal operations and transitions that are not shown represent illegal operations. Illegal operations shall generate exceptions if invoked. The statecharts do not include MOM or DDM activities.

The Normal Activity Permitted state depicted in Figure 3 is elaborated in further detail in Figure 4, to identify the major portions of joined federate state. Additionally, the states in Figure 4 are further elaborated as follows:

- Temporal state (Figure 16)
- States associated with each object class (Figure 9)
- States associated with each interaction class (Figure 12)

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 Normal Activity Permitted). Time management is elaborated in further detail in Figure 16. The state of an arbitrary object class is elaborated in further detail in Figure 9, and the state of an arbitrary interaction class is elaborated in further detail in Figure 12. Indices, such as “i” in Object Class (i), are used in the names of many statecharts to denote that the statechart refers to an arbitrary occurrence of something. These indices are used consistently throughout the 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* sth joined federate; also used as F_s

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

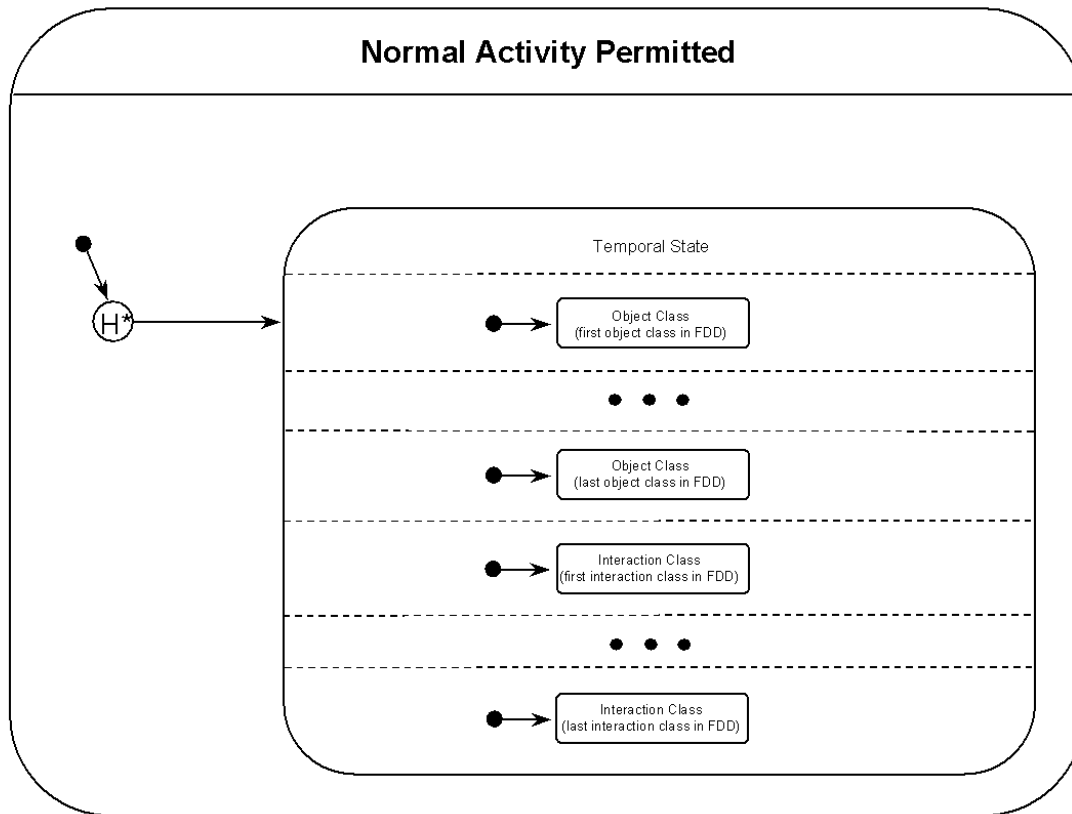


Figure 4—Normal activity permitted

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 time stamp 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* $\dagger$ service at all of these joined federates as soon as possible. If there is a time stamp argument provided, the RTI will invoke the *Initiate Federate Save* $\dagger$ 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* $\dagger$ 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* $\dagger$ 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.

The saved information shall be persistent, meaning that 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 thus 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 that they were in when the save was accomplished. The federation can then resume execution of the execution from that saved point. 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 5 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 elaborated. Because multiple joined federates are involved, each transition within this set of statecharts 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 6. Not shown in this diagram is how the RTI calculates the synchronization set (shown as *synch_set* within the statechart); this information is elaborated in the service descriptions. Use of the synchronization set, however, is depicted in the statechart 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 only achieved 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 elaborated in Figure 7.

The final statechart in the set describing synchronization points is Figure 8. This statechart 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.

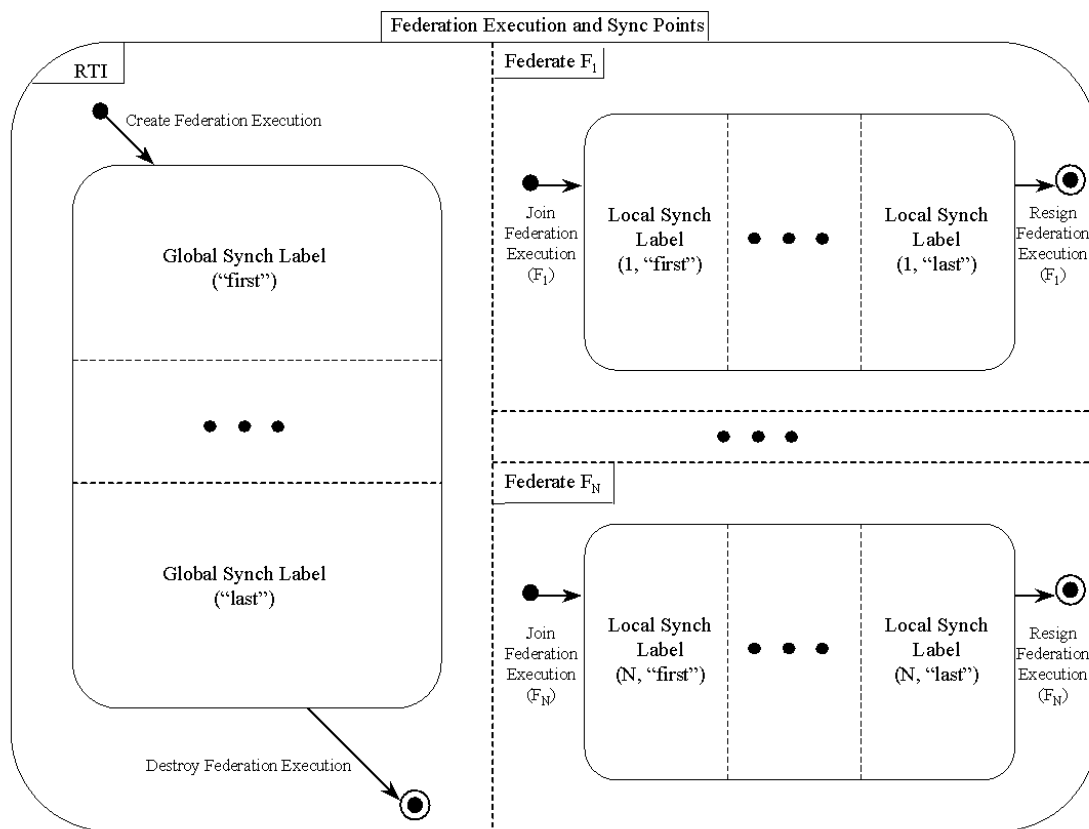


Figure 5—Federation execution and sync points

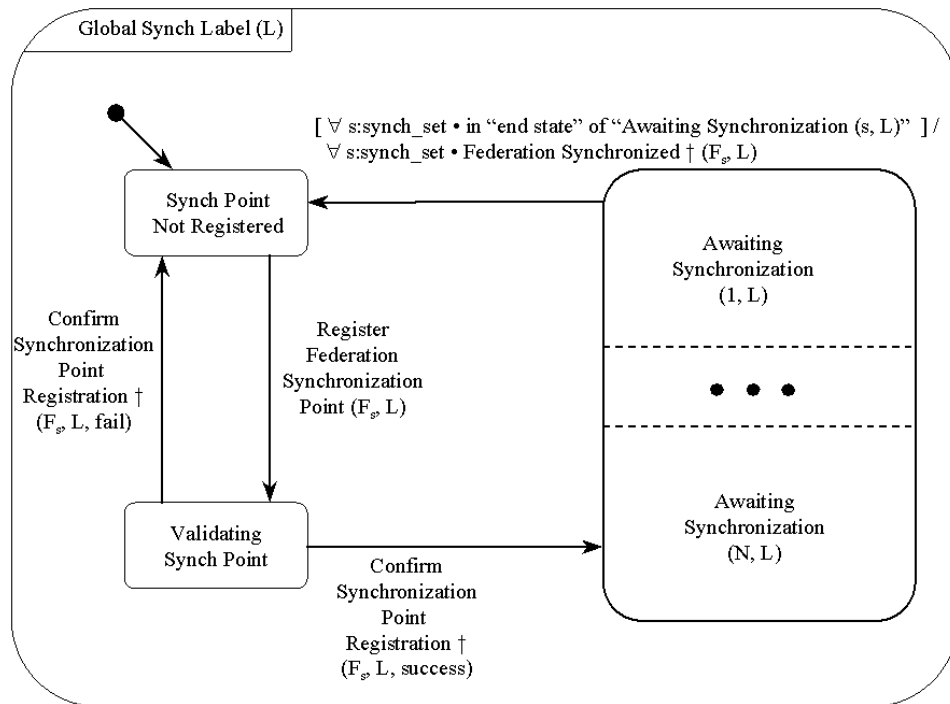


Figure 6—Global synch label (L)

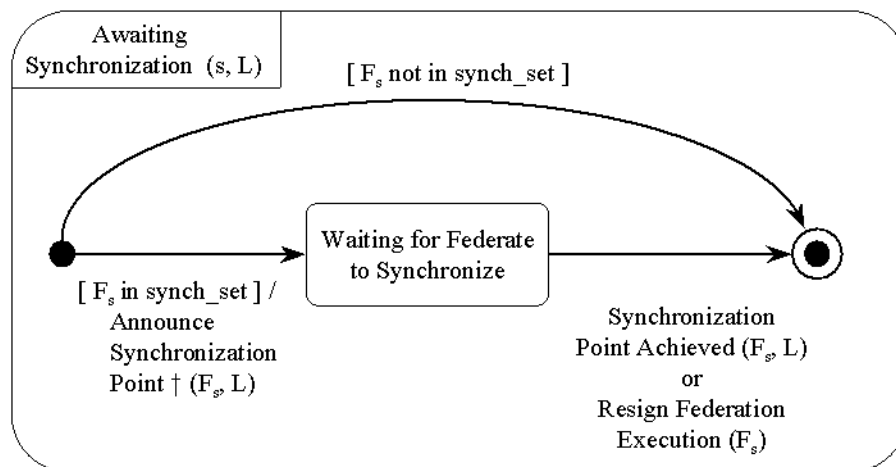


Figure 7—Awaiting synchronization (s, L)

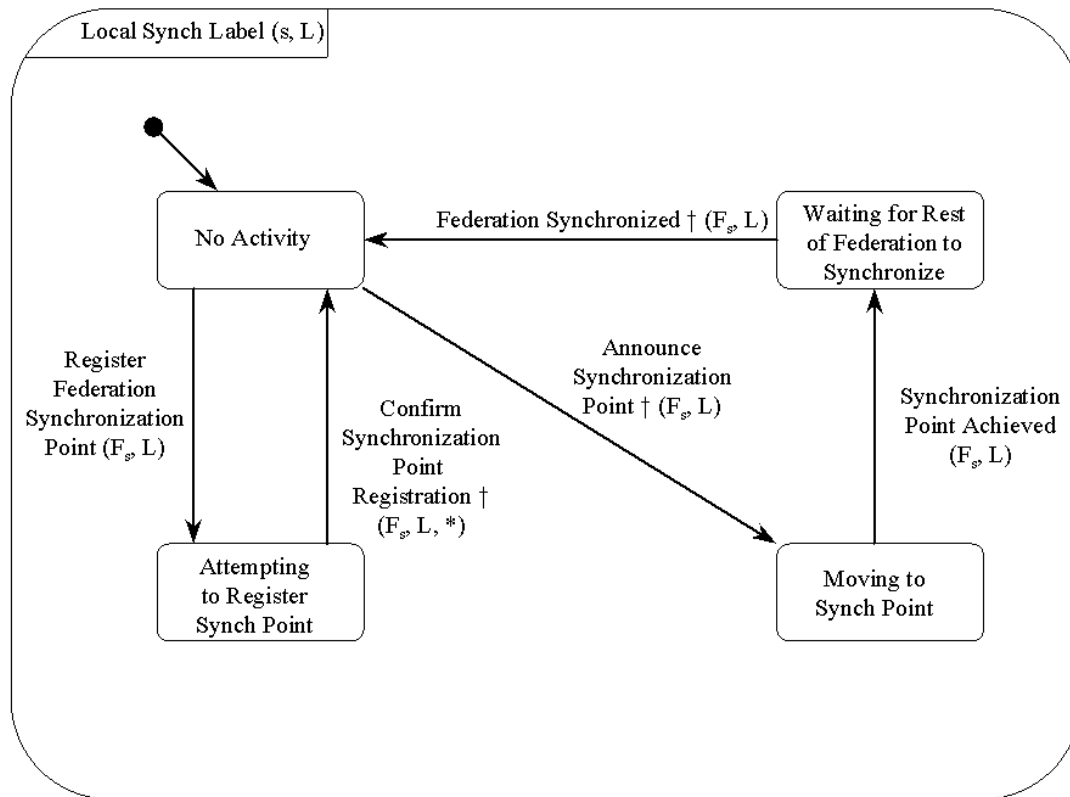


Figure 8—Local synch label (s, L)

4.1.4 FOM document data (FDD)

The FDD is that data and information in a FOM document, as defined in IEEE Std 1516.2-2000, that is used by the *Create Federation Execution* service to initialize a newly created federation execution. The FOM document designator argument supplied to that service shall be a compliant FOM document (IEEE Std 1516-2000 and IEEE Std 1516.2-2000). *Create Federation Execution* shall reject, issuing an exception, any entity that is not a compliant FOM document. In order to support functionality described in this specification, the FDD that the RTI uses from the FOM document shall be, at a minimum, the following:

- Object class structure table
- Interaction class structure table
- Attribute table (transportation, order, and available dimensions columns only)
- Parameter table (transportation, order, and available dimensions columns only)
- Dimension table
- Transportation type table
- Switches table

4.2 Create Federation Execution

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 FOM document designator argument shall identify the FOM that furnishes the FDD for the federation execution to be created.

4.2.1 Supplied arguments

- a) Federation execution name
- b) FOM document designator

4.2.2 Returned arguments

- a) None.

4.2.3 Preconditions

- a) The federation execution does not exist.

4.2.4 Postconditions

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

4.2.5 Exceptions

- a) The specified federation execution already exists.
- b) Could not locate FOM document indicated by supplied designator.
- c) Invalid FOM document.
- d) RTI internal error.

4.2.6 Reference state charts

- a) Figure 1: Basic states of the federation execution.
- b) Figure 5: Federation execution and sync points.

4.3 Destroy Federation Execution

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.

4.3.1 Supplied arguments

- a) Federation execution name.

4.3.2 Returned arguments

- a) None

4.3.3 Preconditions

- a) The federation execution exists.
- b) There are no federates joined to this federation execution.

4.3.4 Postconditions

- a) The federation execution does not exist.

4.3.5 Exceptions

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

4.3.6 Reference state charts

- a) Figure 1: Basic states of the federation execution.
- b) Figure 5: Federation execution and sync points.

4.4 Join Federation Execution

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-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.

4.4.1 Supplied arguments

- a) Federate type.
- b) Federation execution name.

4.4.2 Returned arguments

- a) Joined federate designator.

4.4.3 Preconditions

- a) The federation execution exists.
- b) The federate is not joined to that federation execution.
- c) Save not in progress.
- d) Restore not in progress.

4.4.4 Postconditions

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

4.4.5 Exceptions

- a) The federate is already joined to the federation execution.
- b) The specified federation execution does not exist.
- c) Save in progress.
- d) Restore in progress.
- e) RTI internal error.

4.4.6 Reference state charts

- a) Figure 1: Basic states of the federation execution
- b) Figure 3: Lifetime of a federate
- c) Figure 5: Federation execution and sync points

4.5 Resign Federation Execution

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 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).

4.5.1 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.

4.5.2 Returned arguments

- a) None.

4.5.3 Preconditions

- a) The federation execution exists.
- b) The federate is joined to that federation execution.
- c) If directive 2) is supplied, the joined federate does not own any instance attributes of object instances for which it does not also have the delete privilege.
- d) If directive 3) or 6) is supplied, the joined federate does not own any instance attributes in the federation execution.
- e) If either directive 1), 2), 4), or 6) is specified, there are no instance attributes for which the joined federate has either
 - 1) Invoked the *Attribute Ownership Acquisition* service and has not yet received a corresponding invocation of either the *Confirm Attribute Ownership Acquisition Cancellation* † service or the *Attribute Ownership Acquisition Notification* † service.
 - 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 condition 1) applies]

4.5.4 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) is supplied, all object instances for which the joined federate has the delete privilege are deleted.
- d) If directive 3) or 5) is supplied, all pending instance attribute ownership acquisitions for the joined federate are canceled.

4.5.5 Exceptions

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

4.5.6 Reference state charts

- a) Figure 1: Basic states of the federation execution
- b) Figure 3: Lifetime of a federate
- c) Figure 5: Federation execution and sync points
- d) Figure 7: Awaiting synchronization (s, L)

4.6 Register Federation Synchronization Point

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 used by the joined federate to specify the joined federates in the federation execution that should be informed of the label existence. 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.

4.6.1 Supplied arguments

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

4.6.2 Returned arguments

- a) None

4.6.3 Preconditions

- a) The federation execution exists.
- b) The federate is joined to that federation execution.
- c) If an optional set of joined federate designators is supplied, those federates shall be joined to the federation execution.
- d) The supplied synchronization point label shall not be in use.
- e) Save not in progress.
- f) Restore not in progress.

4.6.4 Postconditions

- a) The synchronization label is known to the RTI.

4.6.5 Exceptions

- a) The federate is not a federation execution member.
- b) Save in progress.
- c) Restore in progress.
- d) RTI internal error.

4.6.6 Reference state charts

- a) Figure 6: Global synch label (L)
- b) Figure 8: Local synch label (s, L)

4.7 Confirm Synchronization Point Registration †

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. If the registration-success indicator argument indicates success, this shall mean the label has been successfully registered.

If the registration-success indicator argument indicates failure, the optional 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.

4.7.1 Supplied arguments

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

4.7.2 Returned arguments

- a) None.

4.7.3 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.

4.7.4 Postconditions

- a) If the registration success indicator is positive, the specified label and associated user-supplied tag will 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.

4.7.5 Exceptions

- a) Federate internal error.

4.7.6 Reference state charts

- a) Figure 6: Global synch label (L)
- b) Figure 8: Local synch label (s, L)

4.8 Announce Synchronization Point †

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, at either 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.

4.8.1 Supplied arguments

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

4.8.2 Returned arguments

- a) None

4.8.3 Preconditions

- a) The federation execution exists.
- b) The federate is joined to that federation execution.
- c) The synchronization point has been registered.

4.8.4 Postconditions

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

4.8.5 Exceptions

- a) Federate internal error

4.8.6 Reference state charts

- a) Figure 7: Awaiting synchronization (s, L)
- b) Figure 8: Local synch label (s, L)

4.9 Synchronization Point Achieved

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.

4.9.1 Supplied arguments

- a) Synchronization point label

4.9.2 Returned arguments

- a) None

4.9.3 Preconditions

- a) The federation execution exists.
- b) The federate is joined to that federation execution.
- c) The synchronization point has been announced at this federate.
- d) Save not in progress.
- e) Restore not in progress.

4.9.4 Postconditions

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

4.9.5 Exceptions

- a) The synchronization point label is not announced.
- b) The federate is not a federation execution member.
- c) Save in progress.
- d) Restore in progress.
- e) RTI internal error.

4.9.6 Reference state charts

- a) Figure 7: Awaiting synchronization (s, L).
- b) Figure 8: Local synch label (s, L).

4.10 Federation Synchronized †

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, indicating that the joined federates in the synchronization set have synchronized at that point. Once the synchronization set for a point synchronizes (the *Federation Synchronized* † service 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.

4.10.1 Supplied arguments

- a) Synchronization point label

4.10.2 Returned arguments

- a) None

4.10.3 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 *Synchronization Point Achieved* using the specified label.

4.10.4 Postconditions

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

4.10.5 Exceptions

- a) Federate internal error

4.10.6 Reference state charts

- a) Figure 6: Global synchlabel (L)
- b) Figure 8: Local synch label (s, L)

4.11 Request Federation Save

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

If the optional time stamp 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 time stamp argument may be supplied only if the joined federate is time-regulating or if GALT is defined for the joined federate. Supplied time stamp 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 only using time stamps that are valid for the joined federate to send in a TSO message (see 8.1 for more information on valid TSO message time stamps). For nonregulating joined federates at which GALT is defined, this can be ensured by only using time stamps that are greater than the joined federate's GALT value.

If a valid optional time stamp 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 time stamps less than or equal to the time stamp of the scheduled save⁶ and either

- The joined federate is in the Time Advancing state as a result of invocation of either 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 time stamp 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 time stamp of the scheduled save

When all time-constrained joined federates are eligible to be instructed to save, the RTI shall instruct all non-time-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 save in progress, which is between the RTI invocation of the *Initiate Federate Save* † service and the RTI invocation of the *Federation Saved* † service.

4.11.1 Supplied arguments

- a) Federation save label
- b) Optional time stamp

4.11.2 Returned arguments

- a) None

4.11.3 Preconditions

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

4.11.4 Postconditions

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

⁶This means that the federate's GALT must be greater than the time stamp of the scheduled save.

4.11.5 Exceptions

- a) Time stamp has already passed (if optional time stamp argument supplied).
- b) Time stamp is invalid (if optional time stamp argument is supplied).
- c) Federate may not use the time stamp argument (if optional time stamp argument is supplied).
- d) The federate is not a federation execution member.
- e) Save in progress.
- f) Restore in progress.
- g) RTI internal error.

4.11.6 Reference state charts

- a) Figure 3: Lifetime of a federate

4.12 Initiate Federate Save †

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 time stamp argument, that time stamp will be provided to the resulting *Initiate Federate Save †* service invocation.

4.12.1 Supplied Arguments

- a) Federation save label
- b) Optional time stamp

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) A federation save has been scheduled.

4.12.4 Postconditions

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

4.12.5 Exceptions

- a) The time stamp is invalid.
- b) Unable to perform save.
- c) Federate internal error.

4.12.6 Reference state charts

- a) Figure 3: Lifetime of a federate

4.13 Federate Save Begun

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

4.13.1 Supplied arguments

- a) None

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 joined federate has received an *Initiate Federate Save* $\dagger$ invocation.
- d) The joined federate is ready to start saving its state.

4.13.4 Postconditions

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

4.13.5 Exceptions

- a) Save not initiated
- b) The federate is not a federation execution member
- c) Restore in progress
- d) RTI internal error

4.13.6 Reference state charts

- a) Figure 3: Lifetime of a federate

4.14 Federate Save Complete

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.

4.14.1 Supplied arguments

- a) Federate save-success indicator

4.14.2 Returned arguments

- b) None

4.14.3 Preconditions

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

4.14.4 Postconditions

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

4.14.5 Exceptions

- a) Invalid save-success indicator.
- b) Federate has not begun save.
- c) The federate is not a federation execution member.
- d) Restore in progress.
- e) RTI internal error.

4.14.6 Reference state charts

- a) Figure 3: Lifetime of a federate

4.15 Federation Saved †

The *Federation Saved* † service shall inform the joined federate that the federation save process is complete, and it shall indicate whether it completed successfully or not. If the save-success indicator argument indicates success, this 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 has 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 has 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 time stamp supplied with the save request cannot be honored (due to possible race conditions in the distributed calculation of GALT).

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.

4.15.1 Supplied arguments

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

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.

4.15.4 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.

4.15.5 Exceptions

- a) Federate internal error

4.15.6 Reference state charts

- a) Figure 3: Lifetime of a federate

4.16 Query Federation Save Status

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

4.16.1 Supplied arguments

- a) None

4.16.2 Returned arguments

- a) None

4.16.3 Preconditions

- a) The federation execution exists.
- b) The federate is joined to that federation execution.
- c) Save in progress.
- d) Restore not in progress.

4.16.4 Postconditions

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

4.16.5 Exceptions

- a) The federate is not a federation execution member.
- b) Save not in progress.
- c) Restore in progress.
- d) RTI internal error.

4.16.6 Reference state charts

- a) None

4.17 Federation Save Status Response †

The *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 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 save in progress
- Federate instructed to save
- Federate saving
- Federate waiting for federation to save

4.17.1 Supplied arguments

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

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.

4.17.4 Postconditions

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

4.17.5 Exceptions

- a) Federate internal error

4.17.6 Reference state charts

- a) None

4.18 Request Federation Restore

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.

4.18.1 Supplied arguments

- a) Federation save label

4.18.2 Returned arguments

- a) None

4.18.3 Preconditions

- a) The federation execution exists.
- b) The federate is joined to that federation execution.
- c) The federation has a save with the specified label.
- d) 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.
- e) All previous *Request Federation Restore* service invocations from the joined federate have been acknowledged with a corresponding *Confirm Federation Restoration Request* †.
- f) Save not in progress.
- g) Restore not in progress.

4.18.4 Postconditions

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

4.18.5 Exceptions

- a) The federate is not a federation execution member.
- b) Save in progress.
- c) Restore in progress.
- d) RTI internal error.