IEEE

1516.2™

IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)— Object Model Template (OMT) Specification

IEEE Computer Society

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

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

18 August 2010

IEEE Std 1516.2™-2010
(Revision of
IEEE Std 1516.2-2000)

IEEE Std 1516.2™-2010
(Revision of
IEEE Std 1516.2-2000)

IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)— Object Model Template (OMT) Specification

Sponsor

Stimulation Interoperability Standards Organization/ Standards Activity Committee (SISO/SAC)
of the
IEEE Computer Society

Approved 25 March 2010

IEEE-SA Standards Board

Abstract: The High Level Architecture (HLA)—Object Model Template (OMT) specification defines the format and syntax (but not content) of HLA object models. Simulations are abstractions of the real world, and no one simulation can solve all of the functional needs for the modeling and simulation community. It is anticipated that advances in technology will allow for new and different modeling and simulation (M&S) implementations within the framework of the HLA. The standards contained in this architecture are interrelated and need to be considered as a product set, as a change in one is likely to have an impact on the others. As such, the HLA is an integrated approach that has been developed to provide a common architecture for simulation.

Keywords: architecture, class attribute, data distribution management, federate, federation, federation execution, federation object model, framework, High Level Architecture, instance attribute, instance attribute ownership, interaction class, joined federate, object class, object model template, rules, runtime infrastructure, simulation object model, time-constrained, time-regulating

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

Copyright © 2010 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 18 August 2010. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers, Incorporated.

PDF: ISBN 978-0-7381-6249-2 STD96060
Print: ISBN 978-0-7381-6250-8 STDPD96060

Introduction

This introduction is not part of IEEE Std 1516.2-2010, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Object Model Template (OMT) Specification.

This document has been developed to record an international standard for the HLA. It serves as one of three related standards for the HLA. It defines the format and syntax for recording information in HLA object models.

This new version of the IEEE 1516 High Level Architecture has been produced during 2004–2007 by the "HLA Evolved Product Development Group" of the Simulation Interoperability Standards Organization (SISO). It incorporates a number of updates based on practical application of earlier versions of the standard. The purpose of the new version is to better support development, deployment, and net-centricity of distributed simulations. Major new additions include support for Web Services communication, modular information models (FOMs and SOMs), improved XML features (XML Scemas as well as extensibility), improved fault tolerance support, support for update rate reduction and dynamic link compatibility between different implementations.

Notice to users

Laws and regulations

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

Copyrights

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

Updating of IEEE documents

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

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

Errata

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

Interpretations

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

Patents

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

Participants

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

The working group acknowledges the following Drafting Group members who provided significant contributions to the preparation of this standard:

Robert Lutz, *OMT Drafting Group Editor*

Jake Borah	Björn Löfstrand	John Schloman
Paul Gustavson	Katherine Morse	Roy Scrudder
Mike Lightner	Chris Rouget	Graham Shanks
Reed Little		Chris Turrell

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

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

Contents

IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)— Object Model Template (OMT) Specification

1. Overview

1.1 Scope

This document defines the format and syntax for recording information in High Level Architecture (HLA) object models, to include objects, attributes, interactions, and parameters. It does not define the specific data (e.g., vehicles, unit types) that will appear in the object models.

1.2 Purpose

The HLA has been developed to provide a common architecture for modeling and simulation. The HLA requires that federations (sets of federates) and individual federates (simulations, supporting utilities, or interfaces to live systems) be described by an object model that identifies the data exchanged at runtime to achieve federation objectives. This standard defines the documentation of the object model.

1.3 Background

1.3.1 Object model template rationale

A standardized structural framework, or template, for specifying HLA object models is an essential component of the HLA for the following reasons:

- It provides a commonly understood mechanism for specifying the exchange of data and general coordination among members of a federation.
- It provides a common, standardized mechanism for describing the capabilities of potential federation members.
- It facilitates the design and application of common tool sets for development of HLA object models.

HLA object models may be used to describe an individual federation member (federate), creating an HLA simulation object model (SOM), or to describe a named set of multiple interacting federates (federation), creating a federation object model (FOM). HLA object models may also be used to define subsets of a FOM or SOM, which are identified as FOM modules and SOM modules respectively. In whichever case, the primary objective of the HLA object model template (OMT) is to facilitate interoperability among simulations and reuse of simulation components. All discussion of HLA object models in this document applies to FOMs, SOMs, FOM modules, and SOM modules, unless explicitly stated otherwise.

1.3.2 Federation object models

During development of an HLA federation, it is critical that all federation members achieve a common understanding as to the nature or character of all required communications among participating federates. The primary purpose of an HLA FOM is to provide a specification for data exchange among federates in a common, standardized format. The content of this data includes an enumeration of all object and interaction classes pertinent to the federation and a specification of the attributes or parameters that characterize these classes. Taken together, the individual components of an HLA FOM establish the "information model contract" that is necessary (but not sufficient) to achieve interoperability among the federates.

A FOM can be constructed as a single self-contained entity, or can be composed from multiple FOM modules. In this first case, the FOM is said to consist of exactly one FOM module. A base object model (BOM) represents one possible method for formulating a FOM module.

1.3.3 Simulation object models

A critical step in the formation of a federation is the process of determining the composition of individual federates to best meet the sponsor's overall objectives. An HLA SOM is a specification of the types of information that an individual federate can provide to HLA federations and 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.

The HLA OMT formats described in this document are generally applicable to either FOMs or SOMs, and also the FOM modules and SOM modules that can be used to formulate a FOM or SOM. Thus, SOMs and SOM modules are also characterized by their objects, attributes, interactions, and parameters. A SOM is specified using one or more SOM modules and one optional Management Object Model (MOM) and Initialization Module (MIM). The primary benefit from the common utilization of the OMT formats for FOMs, FOM modules, SOMs, and SOM modules is that it provides a common frame of reference for describing object models in the HLA community. In some cases, this commonality may even allow SOM modules to be integrated as piece parts in a FOM, facilitating FOM construction. A BOM represents an example of this "piece part" concept and can be used to formulate a SOM module.

1.3.4 Relationship of HLA and object-oriented concepts

Although the HLA OMT is the standardized documentation structure for HLA object models, FOMs, FOM modules, SOMs, and SOM modules do not completely correspond to common definitions of object models in object-oriented 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 object-oriented (OO) techniques recommend defining several views of the system. For HLA object models, the intended scope of the system description is much narrower, focusing mainly on requirements and capabilities for federate information exchange. For SOMs and SOM modules, the intent is to describe the public interface of the federate in terms of an identified set of supported HLA object classes and interaction classes. A more complete description of how a federate is designed and functions internally (for example, a traditional OO object model) should be provided via documentation resources other than the SOM. For FOMs and FOM modules, the intent is to describe information exchange that happens during a federation execution.

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

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

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

Specialization of HLA object and interaction classes are discussed in terms of data tree structures. A data tree structure is defined as, "an abstract hierarchical structure consisting of nodes [classes] connected by branches, in which; each branch connects one node [class] to a directly subsidiary node; there is a unique node called the root [root class] which is not subsidiary to any other node; and every node [class] besides the root is directly subsidiary to exactly one other node [class]" (*The IEEE Standards Dictionary: Glossary of Terms & Definitions*).[1] Furthermore, a node without any subsidiary node is called a leaf node [leaf class].

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]*The IEEE Standards Dictionary: Glossary of Terms & Definitions* is available at http://shop.ieee.org/.

2. Normative references

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

ANSI X3.4-1986 (R1997), Information Systems—Coded Character Sets—7 Bit American National Standard Code Information Interchange (7-Bit ASCII).[2]

IEEE Std 754™-1985 (Reaff 1990), IEEE Standard for Binary Floating-Point Arithmetic.[3, 4]

IEEE Std 1516.1™-2010, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Federate Interface Specification.

REC-XML-20060816, Extensible Markup Language (XML) 1.0 (Fourth Edition), W3C Recommendation, Aug. 2006.[5]

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

3. Definitions, acronyms, and abbreviations

3.1 Definitions

For the purposes of this document, the following terms and definitions apply. *The IEEE Standards Dictionary: Glossary of Terms & Definitions* should be referenced for terms not defined in this clause.[6]

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

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

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

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

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

[2]ANSI publications are available from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA (http://www.ansi.org/).

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

[4]The IEEE standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

[5]This document is available at http://www.w3.org/.

[6]*The IEEE Standards Dictionary: Glossary of Terms & Definitions* is available at http://shop.ieee.org/.

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

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

available dimensions:

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

NOTE—See 9.1 in IEEE Std 1516.1-2010.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

NOTE—See 9.1 in IEEE Std 1516.1-2010.

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

NOTE—See 6.10 in IEEE Std 1516.1-2010.

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

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

NOTE—See 9.1 of IEEE Std 1516.1-2010.

discover: To receive an invocation of the *Discover Object Instance †*[7] service for a particular object instance.

NOTE—See 6.5 of IEEE Std 1516.1-2010.

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 1516.1-2010.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    NOTE—See 6.1 of IEEE Std 1516.1-2010.

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

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

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

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

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

known class:

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

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

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

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

NOTE—See 6.12 of IEEE Std 1516.1-2010.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    NOTE—See 6.1 of IEEE Std 1516.1-2010.

overlap:

a) Pertaining to region sets: given two region sets, to have a region in each set such that the two regions overlap.

b) Pertaining to regions: given two regions, to have at least one dimension in common if and only if, for each dimension the regions have in common, their respective ranges in the common dimension overlap. If two regions do not have any dimensions in common, the regions do not overlap.

c) Pertaining to ranges: given two ranges (A = [a_{lower}, a_{upper}) and B = [b_{lower}, b_{upper})), to overlap if and only if either $a_{\text{lower}} = b_{\text{lower}}$ or ($a_{\text{lower}} < b_{\text{upper}}$ and $b_{\text{lower}} < a_{\text{upper}}$).

NOTE—See 9.1 of IEEE Std 1516.1-2010.

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

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

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

parameter: A named characteristic of an interaction.

passel: A group of attribute handle/value pairs from an *Update Attribute Values* service invocation that are delivered together via a *Reflect Attribute Values †* service invocation. All pairs within the passel have the same user-supplied tag, sent message order type, transportation type, receive message order type, timestamp (if present), and set of sent region designators (if present). A passel is a message.

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

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

NOTE—See 5.1.3 of IEEE Std 1516.1-2010.

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

published:

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

NOTE—See 5.1.2 of IEEE Std 1516.1-2010.

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

NOTE—See 5.1.3 of IEEE Std 1516.1-2010.

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

NOTE—See 5.1.2 of IEEE Std 1516.1-2010.

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 1516.1-2010.

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 1516.1-2010.

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

NOTE—See 9.1 of IEEE Std 1516.1-2010.

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 1516.1-2010.

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 1516.1-2010.

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

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

NOTE—See 6.7 of IEEE Std 1516.1-2010.

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

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

NOTE—See 9.1 of IEEE Std 1516.1-2010.

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 1516.1-2010.

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 1516.1-2010.

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

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

NOTE—See 6.4 of IEEE Std 1516.1-2010.

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

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

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

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

NOTE—See 6.11 of IEEE Std 1516.1-2010.

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

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

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

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

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

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

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

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

NOTE—See 5.1.3 of IEEE Std 1516.1-2010.

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

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

NOTE—See 5.1.3 of IEEE Std 1516.1-2010.

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

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

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

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

NOTE—See 9.1.2 of IEEE Std 1516.1-2010.

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

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

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

subscribed:

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

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

NOTE—See 5.1.2 and 5.6 of IEEE Std 1516.1-2010.

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 1516.1-2010.

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

NOTE—See 5.1.3 and 5.8 of IEEE Std 1516.1-2010.

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

NOTE—See 9.10 of IEEE Std 1516.1-2010.

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

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

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

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

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

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

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

NOTE—See 8.1 of IEEE Std 1516.1-2010.

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

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

NOTE—See 8.1 of IEEE Std 1516.1-2010.

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

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

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

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

NOTE—See 9.1.3 of IEEE Std 1516.1-2010.

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

NOTE—See 6.6 of IEEE Std 1516.1-2010.

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

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

used for sending:

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

NOTE—See 9.1 of IEEE Std 1516.1-2010.

used for subscription of a class attribute:

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

NOTE—See 9.1 of IEEE Std 1516.1-2010.

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 1516.1-2010.

used for subscription of an interaction class:

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

NOTE—See 9.1 of IEEE Std 1516.1-2010.

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 1516.1-2010.

used for update:

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

NOTE—See 9.1 of IEEE Std 1516.1-2010.

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 1516.1-2010.

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

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

3.2 Abbreviations and acronyms

The following abbreviations and acronyms pertain to this standard, IEEE Std 1516-2010, and IEEE Std 1516.1-2010:

ADT	abstract datatype
API	application programmer's interface
BNF	Backus Naur Form
BOM	base object model
DDM	data distribution management
DIF	data interchange format
DM	declaration management
DTD	document type declaration
FDD	FOM Document Data
FEDEP	Federation Development and Execution Process
FOM	federation object model
GALT	Greatest Available Logical Time
HLA	High Level Architecture
JAR	Java Archive
LIS	language-independent specification
LITS	Least Incoming Timestamp
LRC	Local Runtime Component
MIM	MOM and Initialization Module
MOM	Management Object Model
M&S	modeling and simulation
NA	not applicable
OMT	object model template
OO	object oriented
OOAD	object-oriented analysis and design
POC	point of contact
RID	RTI initialization data
RO	receive order
RTI	runtime infrastructure
SOM	simulation object model
TRADT	time representation abstract datatype
TSO	timestamp order
WSDL	Web Service Definition Language
WSPRC	Web Service Provider RTI Component
XML	extensible markup language

3.3 Conventions

Conventions in this standard specify how entries are to be formulated when completing an HLA object model. The following conventions pertain to this standard.

3.3.1 Names

Names in object models shall adhere to eXtensible Markup Language (XML) naming conventions. XML conventions state that names cannot begin with a number or punctuation character, and require that names be constructed from a combination of letters, digits, hyphens, colons, full stops (periods), and underscores with no spaces or other breaking characters (e.g., tabs, carriage returns). In HLA object models, allowable names shall be further restricted as follows:

a) The period (full stop) is reserved for qualifying class names and shall not be included in a user-defined name within an object model. When fully qualified, the class name includes all predecessor class names (separated by periods) beginning at the root and ending at the class name in question.

b) As recommended in the XML specification, the colon character shall not be used.

c) Names beginning with the string “hla,” or any string that would match (('H'|'h') ('L'|'l') ('A'|'a')), are reserved and shall not be included in user-defined names.

d) The case of all textual data within HLA object models (including names) shall be significant; all textual data defined in an object model document shall be case-sensitive.

e) A name consisting of the string “na,” or any string that would match (('N'|'n') ('A'|'a')), is reserved to indicate that a name is not applicable in this circumstance and shall not be included as a user-defined name.

These rules apply to the following names in HLA object models:

- Object class names
- Interaction class names
- Attribute names
- Parameter names
- Datatype names
- Enumerated datatype enumerators
- Fixed record field names
- Variant record alternative names
- Basic data representation names
- Dimension names
- Transportation type names
- Synchronization point labels
- Note identifying labels

4. HLA OMT components

HLA object models are composed of a group of interrelated components specifying information about classes of objects and their attributes and interactions and their parameters. The information content of these components can be represented in many different ways or presentations. A presentation is the formatting of the information contained in the object model in a particular manner for a particular purpose. For example, the OMT tabular format is designed for presentation on a printed page, whereas the OMT data interchange format (DIF) is a presentation designed for passing an object model between tools. All HLA object models shall be capable of being presented in both the OMT tabular format and the OMT DIF format. This standard defines the OMT content and presents it in OMT tabular format throughout.

The OMT consists of the following components:

— *Object model identification table*: To associate important identifying information with the HLA object model
— *Object class structure table*: To record the namespace of all federate or federation object classes and to describe their class-subclass relationships
— *Interaction class structure table*: To record the namespace of all federate or federation interaction classes and to describe their class-subclass relationships
— *Attribute table*: To specify features of object attributes in a federate or federation
— *Parameter table*: To specify features of interaction parameters in a federate or federation
— *Dimension table*: To specify dimensions for filtering instance attributes and interactions
— *Time representation table*: To specify the representation of time values
— *User-supplied tag table*: To specify the representation of tags used in HLA services
— *Synchronization table*: To specify representation and datatypes used in HLA synchronization services
— *Transportation type table*: To describe the transportation mechanisms used
— *Update rate table*: To specify update rate information
— *Switches table*: To specify initial settings for parameters used by the RTI
— *Datatype tables*: To specify details of data representation in the object model
— *Notes table*: To expand explanations of any OMT table item
— *Interface specification services usage table*: To specify the HLA services used by a federate or in a federation
— *FOM/SOM lexicon*: To define all of the objects, attributes, interactions, and parameters used in the HLA object model

All of the OMT components shall be completed when specifying an HLA object model for both federations and individual federates. However, certain tables may be empty or devoid of domain-specific content. For instance, although federations typically support interactions among their federates, some federates (such as a stealth viewer) might not be involved in interactions. In this situation, the interaction class structure table would contain only the single interaction class required by the HLA and the parameter table would be empty in that federate's SOM or supporting SOM module. It is also expected that federates commonly have objects with attributes of interest across the federation; in such cases, these objects and attributes shall be documented. However, a federate or an entire federation may exchange information solely via interactions; in which case, its object class structure table and attribute table would contain only HLA-required data. The specific rules for the applicability of each OMT table are provided in the table descriptions.

The final HLA OMT component, the FOM/SOM lexicon, is essential to help users understand the semantics of the terms used in an HLA object model.

The HLA MOM specifies a designated set of information elements that are associated with federation executions. Implementation of the MOM information elements as specified in IEEE Std 1516.1-2010 provides a mechanism for management of federation executions using existing HLA services. Inclusion of the MOM is required for all FOMs and is achieved by incorporating the MOM data using a MOM and MIM. In addition, federates that have an inherent capability to interact with or extend the MOM shall include the MOM in their SOM. Inclusion of the MOM means the inclusion of a MIM that contains all tables in Clause 11 of IEEE Std 1516.1-2010, along with any applicable MOM extensions.

The last row in any OMT table is labeled "Note." This allows the object model developer to associate one or more notes with the entire table. If no note is being provided for that table, a value of "NA" should be entered. Although the "Note" row is not explicitly identified or discussed in the table descriptions, it is shown in the table examples for illustration purposes. The notes feature is fully explained in 4.14.

A FOM or SOM is specified using one or more FOM or SOM modules. For a FOM, a MIM is also required whereas for a SOM, inclusion of a MIM depends on federate capabilities with respect to the MOM. Any FOM, SOM, FOM module, or SOM module that fully conforms to all of the rules and constraints stated in this specification is a *compliant object model*. A FOM or SOM module is compliant if it conforms to all rules and constraints for a compliant object model as well as all rules and constraints that apply to FOM or SOM modules. A MIM is compliant if it conforms to all rules and constraints of a compliant object model and contains at least the information provided in the standard MIM (see IEEE Std 1516.1-2010).

The basics of each OMT component are presented in the following separate subclauses. The template format for each component is provided and described, and criteria are suggested to help guide decisions on when to include specific federate or federation features within each of these components for a specific HLA object model. Examples of the usage of each OMT component are also provided; these examples are for illustrative purposes only, and they are not meant to imply any additional requirements. Clause 7. specifies the rules and principles for merging FOM modules and SOM modules. Annex A specifies the notations used in describing OMT tables.

4.1 Object model identification table

4.1.1 Purpose/background

A key design goal for all HLA object models is to facilitate reuse. HLA object models provide information that enables inferences to be drawn regarding the reuse potential of individual federates for new applications. Reuse can also occur at the level of the object model. An existing FOM or FOM module may provide a foundation for the development of new FOMs, or SOMs or SOM modules may be merged to form new FOMs. In either case, to expedite reuse, it is important to include a minimum but sufficient degree of descriptive information in the object model. For instance, when federation developers wish to pose detailed questions about how a federate or federation was constructed, point of contact (POC) information within an HLA object model is extremely important. The purpose of this table is to document certain key identifying information within the object model description.

4.1.2 Table format

The information that identifies HLA object models shall be provided in a simple two-column table. The first column (Category) specifies the categories of data that shall be provided in this table. The second column (Information) shall specify the required information. Entries shall be provided for all rows. "NA" shall be entered in any row for which no information is appropriate. Definitions of the categories, descriptions, and occurrences are shown in Table 1.

Table 1—OMT metadata descriptions

Category	Description	Occurs
Name	This field shall specify the name assigned to the object model.	1
Type	*Type*: This field shall specify the type that the object model represents; valid values are as follows: — *FOM*: the object model describes a federation — *SOM*: the object model describes a federate	1

Table 1—OMT metadata descriptions *(continued)*

Category	Description	Occurs
Version	This field shall specify the version identification assigned to the object model.	1
Modification date	This field shall specify the latest date on which this version of the object model was created or modified. The modification date shall be specified in the format "YYYY-MM-DD" (e.g., 1999-04-15).	1
Security classification	This field shall specify the security classification of the object model. The following values may be used for this field, although other values are also valid: — Unclassified — Confidential — Secret — Top Secret	1
Release restriction	This field shall contain any restrictions on the release of the object model to specific organizations or individuals.	0..many
Purpose	This field shall specify the purpose for which the object model was developed.	0..1
Application domain	This field shall specify the type or class of application to which the object model applies. The following values may be used for this field, although other values are also valid: — Analysis — Training — Test and Evaluation — Engineering — Acquisition	0..1
Description	This field shall provide an account of the content of the object model.	1
Use limitation	This field shall identify any known constraints on the application of this object model.	0..1
Use history	This field shall identify any past applications of the object model.	0..many
Keyword	The two subfields of this field shall collectively identify the key-words that are associated with the object model.	0..many
Taxonomy	This field shall specify the source of the keyword vocabulary.	0..1
Keyword value	This field shall specify the word or concept that is addressed by the object model.	1
POC	The subfields of this field shall collectively identify the necessary point of contact (POC) information for this object model.	1..many

Table 1—OMT metadata descriptions *(continued)*

Category	Description	Occurs
POC type	This subfield shall specify the role that the POC has with respect to the object model. The following values may be used for this field, although other values are also valid: — Primary author — Contributor — Proponent — Sponsor — Release authority — Technical POC	1
POC name	This subfield shall specify the name of the object model POC, including an honorific (e.g., Dr., Ms) or rank, first name, and last name where appropriate.	0..1
POC organization	This subfield shall specify the organization with which the POC is affiliated.	0..1
POC telephone	This subfield shall specify the telephone number for the POC, including the international telephone code for the POC's country.	0..many
POC e-mail	This subfield shall specify the e-mail address of the POC.	0..many
References	This field shall specify pointers to additional sources of information. If the object model is intended to represent a FOM module or SOM module, then this field shall characterize the module as "standalone" or as having a "dependency" upon another module. If the object model represents a composition of existing modules, then this field shall identify references to each FOM module or SOM module component.	0..many
Type	This subfield shall specify the form of the reference material. For FOM or SOM modules, the following predefined values may be used for this field: — Standalone — Dependency — Composed From Otherwise, the general type of reference source should be indicated, such as "Text Document," "Spreadsheet," or "Powerpoint File."	1
Identification	The following conditions shall be met: — If the Reference Type subfield = "Standalone," this subfield shall contain the value "NA". — If the Reference Type subfield = "Dependency," this subfield shall identify the name of all dependent FOM modules or SOM modules. — If the Reference Type subfield = "Composed From," this subfield shall identify the names of all FOM modules or SOM modules that have been merged to form the current object model. If the Reference type subfield is not equal to any of the three predefined values, this subfield shall specify the location of the reference source [e.g., uniform resource identifier (URI), ISBN].	1
Other	This field shall specify other data deemed relevant by the author of the object model.	0..1

Table 1—OMT metadata descriptions *(continued)*

Category	Description	Occurs
Glyph	This field holds the image that can be used to visually represent a FOM or SOM.	0..1
Type	This subfield holds the image type being represented. If the glyph is being referenced rather than provided, a value of "URL" shall be provided.	1
Alt	This subfield provides alternative text if the image represented in the *Glyph* field cannot be displayed. If the Glyph Type subfield = "URL," the URL address shall be provided in this subfield.	0..1
Height	This subfield shall represent the pixel height of the glyph image provided in the *Glyph* field.	0..1
Width	This subfield shall represent the pixel width of the glyph image provided in the *Glyph* field.	0..1

The concept of FOM modules and SOM modules is supported via the "References" field of this table. "Dependent" modules in this context are those that contain references to data in other module(s), such as object class definitions or datatype definitions. "Standalone" modules do not contain such references. Table 2 provides the structure to be used to describe the Table 1 metadata elements.

Table 2—Object model identification table

Category	Information
Name	<name>
Type	<type>
Version	<version>
Modification Date	<date>
Security Classification	<security classification>
Release Restriction	[<release restriction>]
Purpose	[<purpose>]
Application Domain	[<application domain>]
Description	<description>
Use Limitation	[<limitation>]
Use History	[<history>]
Keyword	
Taxonomy	[<taxonomy>]
Keyword Value	[<keyword value>]
POC	
POC Type	<poc type>
POC Name	[<poc name>]
POC Organization	[<poc organization>]
POC Telephone	<poc telephone>
POC E-mail	<poc email>
References	
Type	[<ref type>]
Identification	[<identification>]
Other	<other>
Glyph	[<glyph>]
Type	<type>
Alt	[<alt>]
Height	[<height>]
Width	[<width>]

4.1.3 Inclusion criteria

The categories of information specified in this table shall be included for all HLA FOMs, FOM modules, SOMs, and SOM modules.

4.1.4 Example

Table 3 shows a simple example of the object model identification table. In this example, the *Purpose* entry is shorter (by design) than entries that would be found in this table for most practical applications. The subclauses that follow provide examples of each OMT table for the notional SOM identified in this table.

Table 3—Object model identification table example

Category	Information
Name	RestaurantExample
Type	SOM
Version	1.0 Alpha
Modification Date	1998-01-01
Security Classification	Unclassified
Release Restriction	NA
Purpose	To define an HLA object model for a restaurant federate
Application Domain	Restaurant operations
Description	This object model is intended to define the external HLA interface of the “Joe’s Place” restaurant simulation.
Use Limitation	NA
Use History	Federated Foods federation
Keyword	
Taxonomy	Food Service Industry Taxonomy
Keyword	Restaurant, Menu, Waiter
POC	
POC Type	Sponsor
POC Name	Mr. Joseph Doe
POC Organization	Joe’s Place
POC Telephone	1-977-555-1234
POC Email	doej@joesplace.com
References	
Type	HTML document
Identification	www.fedfoods.com/restsim.html
Other	See Mobil International Restaurant Guide for more information
Glyph	🍴

Table 3—Object model identification table example *(continued)*

Category	Information
Type	GIF
Alt	Restaurant
Height	32
Width	32
Note	NA

4.2 Object class structure table

4.2.1 Purpose/background

The object class structure of an HLA object model is defined by a set of relations among classes of objects from the simulation or federation domain. An HLA object class is a collection of objects with certain characteristics or attributes in common. Each of the individual objects that are realizations of a class is said to be an instance of that class.

An HLA object class structure shall be defined by hierarchical relationships among classes of objects. Class relationships shall be represented by the inclusion of the associated class names in the appropriate columns of the object class structure table. Relationships among classes at nonadjacent levels of a class hierarchy can be derived through transitivity: if A is a superclass of B, and B is a superclass of C, A is also a superclass of C. Superclass and subclass play inverse roles in these relations: if A is a superclass of B, B is a subclass of A.

Subclasses can be considered to be specializations, or refinements, of their superclasses. Subclasses shall always inherit the attributes of their superclasses, and they may possess additional attributes to provide the desired specialization. For instance, suppose a "Circle" class is defined as a subclass of a "Figure" class. Attributes defined for the Figure class are those that apply to all types of figures, such as "Color" or "Line Thickness." The Circle subclass automatically inherits the attributes of the Figure class; however, an additional "Radius" attribute might also be defined that is only appropriate for this special type of figure.

A class is a leaf of a class structure if it has no subclasses. The object class "HLAobjectRoot" shall be a superclass of all other object classes in a FOM, FOM module, SOM, or SOM module. The HLA object model shall support only single inheritance; in this mechanism, each class has at most one immediate superclass.

HLA defines the following three types of object class attributes:

- **declared attributes:** The set of class attributes of an object class that are associated with that object class in the object class hierarchy tree.
- **inherited attributes:** The set of class attributes of an object class that was declared in a superclass of that object class in the object class hierarchy tree.
- **available attributes:** The set of declared attributes of an object class in union with the set of inherited attributes of that object class.

Federates participating in a federation execution may subscribe to attributes of object classes at any level of the class hierarchy. Conditions and circumstances surrounding discovery and reflection of instance attributes are described in IEEE Std 1516.1-2010.

Object classes provide the means for federation participants to subscribe to information about all individual instances of HLA objects with common characteristics, such as all sports cars or all fighter jets. Classes are also essential to specifying characteristics (attributes) of simulation objects, because these are defined relative to classes of objects, not unique to individual instances. In addition, because basic HLA services (as described in IEEE Std 1516.1-2010) support subscriptions to object class attributes by federates participating in a federation execution, the RTI requires knowledge of all object classes and their attributes if it is to support distribution of HLA object information by class to the federates of a federation execution.

A class hierarchy expands the capabilities of a flat classification scheme by enabling federates to subscribe to information about broad superclasses of objects, such as all ground vehicles, all air vehicles, or all sea vehicles. IEEE Std 1516.1-2010 supports subscription to attributes of any class in an object class hierarchy so that federates can easily subscribe to attributes of all or only those classes of interest. An object class hierarchy also supports simplification of the specification of attributes by placing common attributes of multiple subclasses in a common superclass. Thus, class hierarchies enable simpler management of the interests of different federates in the objects and attributes involved in a federation execution.

4.2.2 Table format

The object class structure template of Table 4 provides a format for representing the class-subclass hierarchy of object classes. It shall be populated by first entering the root class of all object classes in the left-most column; this object class is named “HLAobjectRoot.” Then, the most general object classes shall be entered in the next column, followed by all of their immediate subclasses in the next column, and then further levels of subclasses, as required. The number of intermediate columns used depends on the needs of the federate or federation. A federate or federation that uses a deeper hierarchy than illustrated by the template of the table shall add columns as needed. Finally, the most specific object classes shall be specified in the right-most column. Object class names shall adhere to HLA naming conventions (see 3.3.1).

Although individual class names need not be unique, all object classes shall be uniquely identifiable in an HLA object model when concatenated (via dot notation) with the names of higher-level superclasses.

Each object class in the object class structure table shall be followed by information on publication and subscription capabilities enclosed in parentheses, as designated in the template using the abbreviated variable name *<p/s>*.

For a SOM or SOM module, valid entries for <*p/s*> shall be as follows:

— *P (Publish)*: The federate is capable of publishing at least one attribute of the object class.

— *S (Subscribe)*: The federate is capable of subscribing to at least one attribute of the object class.

— *PS (PublishSubscribe)*: The federate is capable of publishing at least one attribute and subscribing to at least one attribute of the object class.

— *N (Neither)*: The federate is incapable of either publishing or subscribing to any attributes of the object class.

For a FOM or FOM module, the same entries for <*p/s*> are valid. However, an object class is classified as *Publish* or *Subscribe* in the federation if at least one federate is capable of publishing or subscribing to at least one attribute in the context of the object class.

Classes designated as *Subscribe* or *Neither* can have no registered instances, but they can have subclasses that can be registered.

Table 4—Object class structure table

HLAobject Root (<p/s>)	[<class> (<p/s>)]	[<class> (<p/s>)]	[<class> (<p/s>)]	…	[<class> (<p/s>)]
		[<class> (<p/s>)]	[<class> (<p/s>)]	…	[<class> (<p/s>)]
			⋮	…	⋮
			[<class> (<p/s>)]	…	[<class> (<p/s>)]
		[<class> (<p/s>)]	[<class> (<p/s>)]	…	[<class> (<p/s>)]
			⋮	…	⋮
			[<class> (<p/s>)]	…	[<class> (<p/s>)]
		⋮	⋮	…	⋮
	[<class> (<p/s>)]	[<class> (<p/s>)]	[<class> (<p/s>)]	…	[<class> (<p/s>)]
			[<class> (<p/s>)]	…	[<class> (<p/s>)]
			⋮	…	⋮
	⋮	⋮	⋮	…	⋮

4.2.3 Inclusion criteria

In all HLA object models, any object class that is referenced in any table within the object model shall also be included in the object class hierarchy. The criteria for designing an object class hierarchy for an HLA object model is fundamentally different for individual federates than for federations. The object class structure table of a FOM or FOM module represents an agreement between the federates in a federation on how to classify object classes for the purposes of federation executions. The object class structure table of a SOM or SOM module is a type of advertisement of the classes of objects that the federate can support (as Publish or Subscribe) in potential federations. In neither case does the HLA require specific object classes (other than the required root class) or object class hierarchies to appear in the object class structure table.

For individual federates, it is the intrinsic functionality (expressed as classes of objects) that the federate can offer to future HLA federations—and any object classes whose instances (and associated instance attribute values) may potentially represent useful information if generated by other HLA federates—that defines the content of the object class structure table. The structure of the object class hierarchy is driven by how the federate supports class publication and subscription. Rich, deep SOM class hierarchies can provide federates with a significant degree of flexibility in how they can support and participate in federations in the future.

For federations, it is the subscription requirements of the collective set of simulations participating in the federation that drives the content and structure of the object class hierarchy. Although a set of object classes for the most specific types of entities involved in a federation (e.g., cruiser, refrigerator) may completely satisfy the subscription requirements of some types of HLA applications, additional higher-level object classes would be needed if federates wish to be able to subscribe to HLA object information at higher levels of abstraction (e.g., navy surface ships, kitchen appliances). For a federate to be able to subscribe to HLA object information at a desired level of abstraction, an object class at that level of abstraction shall appear in the object class structure table.

For example, suppose a federation involved air, land, and sea forces of many specific types. If a particular federate did not require notification of the specific types of land vehicles, but did require notification of land vehicles in its area of interest, a suitable class (such as *GroundVehicle*) that contained specific vehicle types as subclasses would be needed to make this possible.

Although classes are clearly needed for all objects of interest in a federation, many alternative class hierarchies can be devised to cover any given set of objects. The demarcations and levels of classes selected for an HLA FOM are the result of the federation development process. Object class hierarchies that already exist for individual FOM modules, SOM modules, and SOMs may be incorporated into a FOM object class hierarchy if they meet the interests of the federation as a whole.

4.2.4 Example

Table 5 provides an example of how the object class structure table can be used to represent a simple system. In this case, the system being represented is a typical neighborhood restaurant. The simulation of this restaurant's operations can be considered to be a potential federate in a larger scale federation, perhaps representing the combined, coordinated operation of a chain of restaurants. The intent of this example is not to specify a complete SOM for this system, but rather to provide partial illustrations of how the OMT tables can be used to capture relevant information about the system.

In this example, a subset of a complete object class hierarchy is shown as consisting of five object classes at the most general level. For this simulation, no class decomposition was necessary for the first three classes. For the fourth class, a single level of decomposition is shown resulting in five leaf classes. For the fifth class, several levels of decomposition are shown to illustrate a partial representation of the restaurant's menu. Some of the deeper levels in this hierarchy could have been modeled as attributes (e.g., *ClamChowder* could have been a leaf class, with an attribute of *Type* to represent the enumerated values of *Manhattan* or *NewEngland*). However, the modeler in this example opted to represent the most specific food types as individual classes.

In this example, most nonleaf classes are designated as *Subscribe* only, whereas the leaf classes are designated as both *Publish* and *Subscribe*. One exception to this is the *Employee* class, which contains several attributes that are only published or subscribed to at the subclass level. Another exception is the *ClamChowder* class, which provides an example of a nonleaf class that can be registered (i.e., has attributes that can be published for that class). Finally, *Manhattan* and *NewEngland* provide examples of publishable leaf classes that do not contain subscribable attributes.

Table 5—Object class structure table example

HLA object Root (N)	Customer (PS)				
	Bill (PS)				
	Order (PS)				
	Employee (N)	Greeter (PS)			
		Waiter (PS)			
		Cashier (PS)			
		Dishwasher (PS)			
		Cook (PS)			
	Food (S)	MainCourse (PS)			
		Drink (S)	Water (PS)		
			Coffee (PS)		
			Soda (PS)		
		Appetizers (S)	Soup (S)	ClamChowder (PS)	Manhattan (P)
					NewEngland (P)
				BeefBarley (PS)	
			Nachos (PS)		
		Entree (S)	Beef (PS)		
			Chicken (PS)		
			Seafood (S)	Fish (PS)	
				Shrimp (PS)	
				Lobster *[Note1] (PS) *[Note2]	
			Pasta (PS)		
		SideDish (S)	Corn (PS)		
			Broccoli (PS)		
			BakedPotato (PS)		
		Dessert (S)	Cake (PS)		
			IceCream (S)	Chocolate (PS)	
				Vanilla (PS)	
Note	NA				

An example of notes has also been included in Table 5. In this case, separate notes have been associated with the class (*Lobster*) and its designation (*PS*). The notes feature is fully explained in 4.14.

4.3 Interaction class structure table

4.3.1 Purpose/background

An interaction is defined as an explicit action taken by a federate that may have some effect or impact on another federate within a federation execution. Interactions shall be specified in the interaction class structure table of HLA object models in terms of their class-subclass relationships, in much the same way that objects are described in the object class structure table. The hierarchical structure of interactions supported by this table is composed of relations of generalization (or specialization) between different types of interactions. For example, an engagement interaction might be specialized by air-to-ground engagements, ship-to-air engagements, and others. This engagement interaction, then, would be said to generalize its more specific types.

An interaction hierarchy in an HLA FOM or FOM module is primarily designed to support inheritance. In SOMs and SOM modules, an interaction hierarchy reflects how the federate supports publication and subscription of interaction classes. The interaction class “HLAinteractionRoot” shall be a superclass of all other interaction classes in a FOM or SOM. The HLA object model shall support only single inheritance; in this mechanism, each class has at most one immediate superclass. Conditions and circumstances surrounding the processing of interactions are described in IEEE Std 1516.1-2010.

Interactions are one of the principal determinants of interoperability among simulations. Interoperability ordinarily requires some consistency in the treatment of interactions afforded by the different federates in which they appear. In distributed war fighting, for example, some uniformity in treatment of engagement interactions is commonly required for a “fair fight” to occur between objects simulated by different federates. Thus, all interactions in a FOM shall be identified and treated (to the extent possible) in a consistent fashion by all federates in the federation.

In addition, the types of interactions involved in a simulation execution shall be made known to the RTI in order to support publication and subscription to their occurrences. Thus, the HLA object model shall document all of the interactions that may be sent during a federation execution so that the RTI can recognize them.

4.3.2 Table format

The template that shall be used for recording HLA interactions for a federation or an individual federate is illustrated in Table 6. The basic format for this table follows the format specified earlier for the object class structure table. Thus, the root interaction class shall be specified in the left-most column and shall be named “HLAinteractionRoot.” Subsequent columns to the right shall contain interaction classes with increasing degrees of class specificity. Like the object class structure table, additional columns may be added to the table as needed to specify the full hierarchy. Although individual class names need not be unique, all interaction classes shall be uniquely identifiable in an HLA object model when concatenated (via dot notation) with the names of higher-level superclasses. Interaction class names shall adhere to HLA naming conventions (see 3.3.1).

Table 6—Interaction class structure table

HLAinteraction Root (<p/s>)	[<class> (<p/s>)]	[<class> (<p/s>)]	[<class> (<p/s>)]	…	[<class> (<p/s>)]
		[<class> (<p/s>)]	[<class> (<p/s>)]	…	[<class> (<p/s>)]
			⋮	…	⋮
			[<class> (<p/s>)]	…	[<class> (<p/s>)]
		[<class> (<p/s>)]	[<class> (<p/s>)]	…	[<class> (<p/s>)]
			⋮	…	⋮
			[<class> (<p/s>)]	…	[<class> (<p/s>)]
		⋮	⋮	…	⋮
	[<class> (<p/s>)]	[<class> (<p/s>)]	[<class> (<p/s>)]	…	[<class> (<p/s>)]
			[<class> (<p/s>)]	…	[<class> (<p/s>)]
			⋮	…	⋮
	⋮	⋮	⋮	…	⋮

Each interaction class in the interaction class structure table shall be followed by information on publishing and subscribing capabilities enclosed in parentheses, as designated in the template using the abbreviated variable name <*p/s*>.

For a SOM or SOM module, valid entries for <*p/s*> shall be as follows:

— *P (Publish)*: The federate is capable of publishing the interaction class.

— *S (Subscribe)*: The federate is capable of subscribing to the interaction class.

— *PS (PublishSubscribe)*: The federate is capable of publishing and subscribing to the interaction class.

— *N (Neither)*: The federate is incapable of either publishing or subscribing to the interaction class.

For a FOM or FOM module, the same entries for <*p/s*> are valid. An interaction class is classified as *Publish* or *Subscribe* in the federation if at least one federate is capable of publishing or subscribing to the class.

Classes designated as *Subscribe* or *Neither* are never sent, but they can have subclasses that are sent.

4.3.3 Inclusion criteria

A type of interaction shall be included in a FOM or FOM module whenever it can take place “across” a federation, i.e., when it is published by one federate and subscribed to by another. Common examples of such interactions in warfighting simulations include a variety of engagement interactions between platforms that may be simulated by different federates. In order to document the types of interactions that federation members may need to accommodate, a FOM shall include all cross-federate interactions.

When interactions are expected to occur within an individual federate, they need not appear in a FOM or FOM module. For example, in an engineering simulation, the interactions involved in the internal dynamics of a vehicle engine should not be part of a FOM if only one federate in the federation interacts directly with the engine component.

Because HLA SOMs and SOM modules are intended to be developed independently of any particular federation application, the relevance of any currently supported interaction class to future federations will generally be unknown. Thus, a simulation that supports either publishing or subscribing for an interaction class should document that support in its SOM and/or relevant SOM modules if it might be of interest to future federations.

4.3.4 Example

A representation of some illustrative interactions, based on the restaurant example introduced in 4.2.4, is given in Table 7. Here, the operations of the restaurant are described according to a set of interactions between customers and the employees of the restaurant. At the highest level, the restaurant federate can publish generic customer-employee transactions. This would, for instance, allow federates that simulate management operations across the restaurant chain to subscribe to and monitor general activity levels for individual restaurants. This single high level interaction class is then decomposed into the basic types of customer-employee interactions that occur in the restaurant. Although all classes at this level can be published by the federate, *OrderTaken*, *FoodServed*, and *CustomerPays* are all further decomposed into more specialized classes that can also be published by the restaurant federate depending on the needs of the federation. In addition, this federate also shows the ability to subscribe to (*S* designation) interactions of classes *CustomerSeated* and *CustomerLeaves*, possibly to monitor customer arrival activity in other restaurants within the chain, and to monitor the rate at which other restaurants can service their customers.

Table 7—Interaction class structure table example

HLAinteractionRoot (N)	CustomerTransaction (P)	CustomerSeated (PS)	
		OrderTaken (P)	FromKidsMenu (P)
			FromAdultMenu (P)
		FoodServed (P)	DrinkServed (P)
			AppetizerServed (P)
			MainCourseServed (P)
			DessertServed (P)
		CustomerPays (P)	ByCreditCard (P)
			ByCash (P)
		CustomerLeaves (PS)	
Note	NA		

4.4 Attribute table

4.4.1 Purpose/background

Each class of simulation domain objects shall be characterized by a fixed set of attribute types. These attributes are named portions of their object's state whose values can change over time (such as location or velocity of a platform). The values of HLA instance attributes are updated through the RTI and provided to other federates in a federation. Both federates and federations shall document all such object attributes in the attribute table of their SOM, FOM, and relevant SOM modules and FOM modules. The object class "HLAobjectRoot" shall be a superclass of all other object classes in a FOM or FOM module and in a SOM or SOM module; attributes may be assigned to it like any other object class.

Attributes of HLA object classes are specified in order to support subscription to their values by other interested members of a federation. Thus, the names of attributes and associated object classes are essential information when initializing a federation execution. Knowledge of object attributes is commonly required for effective communication between federates in a federation. In addition, although the datatypes and update policies of attributes represent characteristics that are not directly utilized by the RTI, they are important to ensuring compatibility among federates in a federation. A federate operating with very low update rates for an instance attribute could create problems for interacting federates that are operating at high update rates. The specification of datatypes and update rates in an HLA FOM is a part of the FOM "contract" among federates to interoperate at the specified levels. These specifications provide a common perception of the simulation space across federates in a federation, lowering the potential for inconsistency.

4.4.2 Table format

The attribute table of a FOM or FOM module shall describe all object attributes represented in a federation. The attribute table of a SOM or SOM module shall describe all class attributes that are published or subscribed to by the federate. The template that shall be used for the attribute table is provided by Table 8.

Table 8—Attribute table

Object	Attribute	Datatype	Update type	Update condition	D/A	P/S	Available dimensions	Transport-ation	Order
HLA object Root	HLA privilege ToDelete Object	HLAtoken	<update type>	<update condition>	<d/a>	<p/s>	<dimensions>	<transport>	<order>
<object class>	<attribute>	<datatype>	<update type>	<update condition>	<d/a>	<p/s>	<dimensions>	<transport>	<order>
	<attribute>	<datatype>	<update type>	<update condition>	<d/a>	<p/s>	<dimensions>	<transport>	<order>
	<attribute>	<datatype>	<update type>	<update condition>	<d/a>	<p/s>	<dimensions>	<transport>	<order>
<object class>	<attribute>	<datatype>	<update type>	<update condition>	<d/a>	<p/s>	<dimensions>	<transport>	<order>
	<attribute>	<datatype>	<update type>	<update condition>	<d/a>	<p/s>	<dimensions>	<transport>	<order>
	<attribute>	<datatype>	<update type>	<update condition>	<d/a>	<p/s>	<dimensions>	<transport>	<order>

The first column (Object) shall contain names from the object class structure table for object classes that are associated with attributes. Object class names shall include the parentage (superclasses) of the class to the depth necessary to uniquely identify the class in this table and may include the full parentage of the class. In general, to reduce redundancy, attributes should be specified for classes at the highest point in the hierarchy to which they apply. For example, if all air vehicles have an attribute of *MinimumTurnRadiusAt MaximumSpeed*, some redundancy can be avoided if this attribute is specified just once for the entire class of *AirVehicle*. Given that all object classes inherit the attribute types of their superclasses, the subclasses of *AirVehicle*, such as *FixedWing* and *RotaryWing,* also have this attribute with its specified characteristics.

The second column (Attribute) shall list the attributes of the specified object class. Attribute names shall adhere to HLA naming conventions (see 3.3.1). The names assigned to attributes of any particular object class shall not duplicate (overload) the names of attributes of this class or any higher-level superclass. There may be many attributes for a single object class; however, all attribute names for a given class shall be unique.

The third column (Datatype) shall identify the datatype of the attribute. This datatype shall be chosen from the name of any simple, enumerated, array, fixed record, or variant record datatype described in subclauses of 4.13. The predefined datatype HLAtoken (see 4.13.6) can be used to indicate that the attribute never contains a value and is used only as a token.

Although an attribute's datatype can be specified "NA," that attribute shall have valid (non-"NA") transportation and order types specified. When a datatype of "NA" is used, the update type, update condition, and available dimensions shall also be "NA."

The fourth column (Update type) shall record the policy for updating an instance of the class attribute. The unique designations for this column shall be as follows:

— *Static*: The value of the attribute is static; the owner of the attribute (federate or RTI) updates it upon registration and when requested.

— *Periodic*: The owner of the attribute (federate or RTI) updates the attribute at regular time intervals.

— *Conditional*: The owner of the attribute (federate or RTI) updates the attribute when unique conditions dictate.

— *NA*: The federate never provides a value for this attribute.

The fifth column (Update condition) shall expand and explain the policies for updating an instance of the class attribute. When the update type is *periodic*, a rate of the number of updates per time unit shall be specified in the Update Condition column. Attributes with a *conditional* update type shall have the conditions for update specified in the "Update condition" column. The "Update condition" column may also be used to specify initial conditions for instance attribute updates. If a federate is capable of varying the rate and circumstances of the update of an instance object attribute, this information shall also be recorded in this column or through the OMT notes feature. If the update type is *Static* or *NA*, "NA" shall be entered in this column.

The sixth column (D/A) shall indicate whether ownership of an instance of the class attribute can be divested or acquired.

In a FOM or FOM module, the ownership of an instance attribute may be designated as divest, acquire, both or neither. Descriptions of the designations for this column are as follows:

— *D (Divest)*: Some federate in the federation is capable of publishing the class attribute and can divest ownership of an instance of this attribute using the HLA ownership management services.

— *A (Acquire)*: Some federate in the federation is capable of publishing the class attribute and can acquire ownership of an instance of this attribute.

— *N (NoTransfer)*: Ownership of an instance of this class attribute is not transferable in this federation.

— *DA (DivestAcquire)*: Some federate in the federation is capable of divesting ownership of instances of this class attribute and another federate is capable of acquiring ownership of instances of this class attribute.

In a SOM or SOM module, the federate may be able to divest ownership of an instance attribute, acquire ownership, both, or neither. Moreover, the corresponding class attribute shall be published for the federate to divest or acquire ownership. Descriptions of the designations for this column are as follows:

— *D (Divest)*: The federate is capable of publishing the class attribute and can divest ownership of an instance of this attribute using the HLA ownership management services.

— *A (Acquire)*: The federate is capable of publishing the class attribute and can acquire ownership of an instance of this attribute.

— *N (NoTransfer)*: The federate is neither capable of divesting ownership of instances of this class attribute nor acquiring ownership of instances of this attribute.

— *DA (DivestAcquire)*: The federate is capable of both divesting ownership of instances of this class attribute and acquiring ownership of instances of this class attribute.

The seventh column (P/S) shall identify the capabilities of a federate or federation with respect to class attribute publishing and subscribing.

In a SOM or SOM module, the entry for this column shall take one of the following values:

— *P (Publish)*: The federate is capable of publishing this class attribute, either in the context of the class where it is declared or in at least one subclass of that class (i.e., capable of invoking appropriate HLA publishing services with the class attribute as a supplied argument).

— *S (Subscribe)*: The federate is capable of subscribing to this class attribute, either in the context of the class where it is declared or in at least one subclass of that class (i.e., capable of invoking HLA appropriate subscribing services with the class attribute as a supplied argument).

— *PS (PublishSubscribe)*: The federate is capable of both publishing and subscribing to this class attribute.

— *N (Neither)*: The federate neither publishes nor subscribes to this class attribute.

In a FOM or FOM module, the same basic designations shall apply.

The eighth column (Available dimensions) shall record the association of the class attribute with a set of dimensions if a federate or federation is using data distribution management (DDM) services for this attribute. The column shall contain a comma-separated list of dimension names corresponding to rows in the dimension table described in 4.6. Class attributes can be individually associated with dimensions, regardless of the dimensions associated with other attributes of the same class. For SOMs and FOMs of federates and federations, respectively, that are not using DDM services or for class attributes that are not associated with dimensions, "NA" shall be entered in this column.

The ninth column (Transportation) shall specify the type of transportation to be used with this attribute. These values shall be chosen from the name of any row in the transportation type table as described in 4.10.

The tenth column (Order) shall specify the order of delivery to be used with instances of this class attribute. Valid values for entries in this column are as follows:

— *Receive*: Instances of the class attribute are delivered to a receiving federate in an undetermined order.

— *TimeStamp*: Instances of the class attribute are delivered to a receiving federate in an order determined by timestamps assigned when the instance attributes were updated.

NOTE—One entry is required for Table 8. For this entry, the first column shall contain "HLAobjectRoot" and the second shall contain "HLAprivilegeToDeleteObject;" subsequent columns shall be completed as appropriate for the federate or federation. Uses and restrictions on this attribute are described in IEEE Std 1516.1-2010.

4.4.3 Inclusion criteria

All instance attributes whose values can be exchanged or ownership can be transferred during the course of an HLA federation execution shall be documented in the attribute table of a FOM and relevant FOM modules. All class attributes that can be either published or subscribed to by an individual federate shall be documented in the attribute table of its SOM and relevant SOM modules.

4.4.4 Example

Table 9 shows examples of attributes from the restaurant application described in 4.2.4. First, the *Employee* object class is characterized according to the four attributes shown in the table. The datatypes specified are defined in the examples provided in 4.13. Each of these four attributes is updated conditionally except for the *YearsOfService* attribute, which is updated periodically (yearly) on the employee's start date anniversary. As with most of the domain-specific attributes shown in this example, the attributes of *Employee* are assumed divestable, acquirable, publishable, and subscribable.

The *Waiter* subclass of *Employee* is shown with three attributes. These are in addition to the four inherited attributes from its superclass. Each of the first two attributes, *Efficiency* and *Cheerfulness*, is intended to represent a performance measure that is assigned to the employee at performance reviews. The third attribute is intended to represent the state of the employee (the task he/she is performing) at any point in time during restaurant operations.

In the final two entries, the *Drink* and *Soda* classes are each shown with a single attribute. The *NumberCups* attribute of the *Drink* class has been associated with the *BarQuantity* dimension, and the *Flavor* attribute of the *Soda* class has been associated with the *SodaFlavor* and *BarQuantity* dimensions; these dimensions are shown in the dimension table example in 4.6.4.

The HLAprivilegeToDeleteObject attribute of class HLAobjectRoot is a different kind of attribute than the others in the table. This attribute has a datatype of "HLAtoken," meaning that an instance of this attribute does not have a value to be updated. Instead, it is more of a token attribute, in which information is conveyed simply by determining which, if any, federate owns the attribute at any time. In this case,

Table 9—Attribute table example

Object	Attribute	Datatype	Update type	Update condition	D/A	P/S	Available dimensions	Transport-ation	Order
HLAobject Root	HLA privilege ToDelete Object	HLAtoken	NA	NA	N	N	NA	HLAreliable	Time-stamp
Employee	PayRate	DollarRate	Conditional	Merit increase *[Note3]	DA	PS	NA	HLAreliable	Time-stamp
	YearsOf Service	Years	Periodic	1/year *[Note4]	DA	PS	NA	HLAreliable	Time-stamp
	Home Number	HLAASCII string	Conditional	Employee request	DA	PS	NA	HLAreliable	Time-stamp
	Home Address	Address Type	Conditional	Employee request	DA	PS	NA	HLAreliable	Time-stamp
Employee. Waiter	Efficiency	Waiter Value	Conditional	Performance review	DA	PS	NA	HLAreliable	Time-stamp
	Cheerful-ness	Waiter Value	Conditional	Performance review	DA	PS	NA	HLAreliable	Time-stamp
	State	Waiter Tasks	Conditional	Work flow	DA	PS	NA	HLAreliable	Time-stamp
Food.Drink	Number Cups	DrinkCount	Conditional	Customer request	N	PS	BarQuantity	HLAreliable	Time-stamp
Food.Drink. Soda	Flavor	FlavorType	Conditional	Customer request	N	PS	SodaFlavor, BarQuantity	HLAreliable	Time-stamp
Note	NA								

HLAprivilegeToDeleteObject is a token attribute known by the RTI and used to determine whether or not a federate is permitted to delete an object instance.

Two examples of notes have also been included in Table 9; this feature is fully explained in 4.14.

4.5 Parameter table

4.5.1 Purpose/background

Most interaction classes are characterized according to a list of one or more interaction parameters. Interaction parameters are used to associate relevant and useful information with classes of interactions. Examples of interaction parameters include object class names, object instance attribute values, strings, and numerical constants. Although some circumstances may require that a subset of the associated parameters be sent with an interaction, other circumstances require that the full set of parameters be sent. However, for every interaction class identified in the interaction class structure table, the full set of parameters associated with that interaction class shall be described in the parameter table. The interaction class "HLAinteractionRoot" shall be a superclass of all other interaction classes in a FOM or FOM module and in a SOM or SOM module; parameters may be assigned to it like any other interaction class.

Unlike class attributes, interaction parameters may not be subscribed to on an individual basis. Thus, dimension, transportation, and delivery order information is specified at the interaction class level rather than at the parameter level.

Interaction parameters may be associated with an interaction class at any level of an interaction class hierarchy. An interaction class inherits the parameters defined for its superclasses. In fact, the mechanisms and rules for inheritance of interaction parameters are identical to those of attributes. Thus, the specific placement of parameters throughout an interaction class hierarchy for a given federation is driven by the same needs that drive the placement of attributes in an object class hierarchy.

4.5.2 Table format

The parameter table of an HLA object model is designed to provide descriptive information about all parameters of all interactions represented in a federation. The template that shall be used for the parameter table is provided in Table 10.

Table 10—Parameter table

Interaction	Parameter	Datatype	Available dimensions	Transportation	Order
<interaction class>	<parameter>	<datatype>	<dimensions>	<transportation>	<order>
	<parameter>	<datatype>			
	<parameter>	<datatype>			
<interaction class>	<parameter>	<datatype>	<dimensions>	<transportation>	<order>
	<parameter>	<datatype>			
	<parameter>	<datatype>			

The first column (Interaction) shall contain names of interaction classes from the interaction class structure table. Interaction class names shall include the parentage (superclasses) of the class to the depth necessary to uniquely identify the class in this table and may include the full parentage of the class. In general, to reduce redundancy, parameters should be specified for classes at the highest point in the hierarchy in which they represent useful information, although this is not required. For example, if all weapon firings include a parameter that defines the infrared signature of the platform at the time the firing occurs, some redundancy would be avoided if this parameter is specified just once at the uppermost level of a *WeaponFires* class. Given that all interaction subclasses inherit the parameter types of their superclasses, the subclasses of *WeaponFires*, such as *TankFires* and *Shipfires,* also have this parameter with its specified characteristics. Note that an interaction class need not contain any parameters; however, all interaction classes shall be included in this table so that transportation and order can be specified.

The second column (Parameter) shall list the parameters of each interaction. Parameter names shall adhere to HLA naming conventions (see 3.3.1). The names assigned to parameters of any particular interaction class shall not duplicate (overload) the names of parameters of this class or any higher-level superclass. There may be many parameters for a single interaction class; however, all parameter names for a given class shall be unique.

If no parameters are specified for an interaction class, "NA" shall be entered in this column.

The third column (Datatype) shall identify the datatype of the parameter. This datatype shall be chosen from the name of any simple, enumerated, array, fixed record, or variant record datatype described in 4.13. If the interaction class has no parameters, "NA" shall be entered for this column.

The fourth column (Available dimensions) shall record the association of an interaction class with a set of dimensions if the federate or federation is using DDM services for this interaction. The column shall contain a comma-separated list of dimension names corresponding to rows from the dimension table described in 4.6. This indicates that whole interactions of this class are subject to filtering through regions containing a subset of these dimensions when the interaction is sent. The available dimensions of an interaction class shall be a superset of the available dimensions of its immediate superclass to ensure that interactions of the specified class are received by subscribers to the superclass when interactions are promoted to the superclass. For SOMs and FOMs in which DDM services are not used or for interactions that are not associated with dimensions, "NA" shall be entered in this column.

The fifth column (Transportation) shall specify the type of transportation to be used with this interaction. These values shall be chosen from the name of any row in the transportation type table as described in 4.10.

The sixth column (Order) shall specify the order of delivery to be used with this interaction. Valid values for entry in this column are as follows:

— *Receive*: The interaction is delivered to a receiving federate in an undetermined order.

— *TimeStamp*: The interaction is delivered to a receiving federate in an order determined by a timestamp assigned when the interaction was initiated.

4.5.3 Inclusion criteria

In federations, any information elements that can be provided and associated with a given interaction class (by the class publishers) that are deemed to be useful to subscribers to that interaction class shall be included and documented as interaction parameters in the parameter table. For individual federates, SOM developers should associate with their publishable interaction classes whatever information they feel could be needed by the subscribers to their interaction classes. In addition, for interaction classes to which the federate can subscribe, SOM developers should determine what types of information need to be included with those interactions so that the federate can calculate associated effects.

4.5.4 Example

Table 11 shows a partial example of interactions and parameters chosen from the restaurant application described in 4.2.4. Here, the *FoodServed.MainCourseServed* interaction has three parameters associated with it. The third parameter uses a Boolean datatype to reflect whether the meal was served in a reasonable amount of time. In addition, this interaction has been associated with the *WaiterId* dimension from the dimension table example in 4.6.4 as a means of filtering on waiter identification numbers. *CustomerSeated* is an example of an interaction with no parameters; the sending and the receiving of the interaction is sufficient to convey the necessary information.

Table 11—Parameter table example

Interaction	Parameter	Datatype	Available dimensions	Transportation	Order
CustomerSeated	NA	NA	NA	HLAreliable	TimeStamp
FoodServed.MainCourse Served	TemperatureOk	ServiceStat	WaiterId	HLAreliable	TimeStamp
	AccuracyOk	ServiceStat			
	TimelinessOk	HLAboolean			
Note	NA				

4.6 Dimension table

4.6.1 Purpose/background

Federations use declaration management (DM) services to enable the flow of instance attribute and interaction data between federates and to limit the delivery of some data on the basis of object class, interaction class, and attribute name. This reduction of data may be insufficient to meet the needs of federations with large numbers of federates, large numbers of objects or interactions in classes, or large numbers of instance attribute updates per object. Such federations can use DDM services to further reduce the volume of data delivered to federates. When DDM services are used by a federation, a common framework for specifying the data distribution model for the federation is essential for the following reasons:

— It provides a commonly understood mechanism for specifying the exchange of public data and DDM coordination among members of a federation.

— It facilitates the design and application of common tool sets for specification of federation DDM needs.

Dimensions are the most fundamental DDM concept. Each attribute or interaction with which DDM services are used shall have a set of available dimensions, as indicated in the attribute table or parameter table, respectively. Each set of available dimensions is a subset of all dimensions that are defined in the dimension table, and defines a multidimensional coordinate system through which federates either express an interest in receiving data or declare their intention to send data. These intentions are specified by creating regions within the coordinate system that indicate particular areas of interest. The regions associated with an instance attribute or interaction shall contain only dimensions that are a subset of the attribute's or interaction's available dimensions, but need not contain all of the available dimensions. These regions are then used in one of the two following ways:

— *Subscription regions*: Sets of ranges that narrow the scope of interest of the subscribing federate in object class attributes and interactions.

— *Update regions*: Sets of ranges that advertise a publishing federate's ability to match subscription interests.

During development of an HLA federation that uses DDM, it is critical that all federation members achieve a common understanding of DDM dimensions and their semantics, and that they agree to a common set of dimension specifications. These agreements are necessary for federates to filter instance attribute updates and interactions in a correct and consistent manner. This shall include the names of dimensions, the association of attributes and interactions with dimensions, the federate view of each dimension, and the default ranges the RTI should supply for available dimensions of attributes and interactions when a federate does not specify ranges at region creation time.

The federate view of each dimension is different from the RTI view of each dimension. The federate view gives each dimension user-defined semantics and an associated datatype in which values are expressed. The federate shall provide a normalization function for each dimension, that maps values from the federate view of a dimension to values in the RTI view of a dimension. Note that the federate view of a dimension is the same for each federate in a federation.

The RTI view of a dimension is an interval of nonnegative integers. The lower bound is fixed for all dimensions, but the upper bound varies by dimension as indicated in the dimension table. The resolution of the RTI view of the dimension does not have to be the same as the resolution of the federate view of the same dimension.

The dimensions form the basis by which update and subscription regions are specified by the federates to the RTI. By having this agreement about the meaning of dimensions enforced at the federate level, the RTI can calculate the overlaps of update and subscription regions efficiently without having to understand the semantics of the dimensions. The dimension table shall record all of the elements necessary to specify this agreement in a standard format.

4.6.2 Table format

The template that shall be used for recording dimensions is illustrated in Table 12.

Table 12—Dimension table

Name	Datatype	Dimension upper bound	Normalization function	Value when unspecified
<dimension>	<type>	<bound>	<normalization function>	<default range/excluded>
<dimension>	<type>	<bound>	<normalization function>	<default range/excluded>
<dimension>	<type>	<bound>	<normalization function>	<default range/excluded>

The first column (Name) shall specify the name of the dimension. Dimension names shall adhere to HLA naming conventions (see 3.3.1). The names of dimensions in this column shall be unique. Each dimension represents a specific characteristic that may be used for filtering. For example, a dimension named *AltitudeLimits* might indicate the desire to filter information based on altitude.

The second column (Datatype) shall identify the datatype for the federate view of the dimension. This datatype shall be chosen from the name of a simple datatype or an enumerated datatype, as described in 4.13.

The third column (Dimension upper bound) shall specify the upper bound for the dimension that meets the federation's requirement for dimension subrange resolution. The value for this column shall be a positive integer. This value limits the resolution the RTI provides to meet the federation's filtering requirements. It is also the maximum value a federate may use when communicating a range in this dimension to the RTI. For example, if a dimension represents radio frequencies in the range 88.1 MHz to 107.3 MHz in increments of .2, a dimension upper bound of 97 is sufficient where 88.1 is mapped to 0 and 107.3 is mapped to 96 in the RTI view.

The fourth column (Normalization function) shall specify the map from a subscription/update region's bounding coordinates to nonnegative integer subranges in the range [0, dimension upper bound). Because regions communicated with an RTI are expressed by bounding coordinates expressed in values from an RTI's view of dimensions, normalization functions shall be used by federates to construct regions. For each dimension in a region, that dimension's normalization function shall be used to map the dimension bounds as expressed in the federate view of the dimension to subranges of [0, dimension upper bound). To ensure that the federation exhibits correct semantics, federation participants should agree on the normalization functions that they intend to use for each dimension. This agreement shall include the specification of the normalization function and all values that are to be used by the function. If an attribute or parameter is used in the normalization function, its class hierarchy shall be specified in dot notation to the extent necessary to identify the attribute or parameter uniquely. The normalization function shall specify whether it uses a subset of the entire range or a subset of the datatype that is the basis of the dimension type. Example normalization functions are listed in Annex B.

The fifth column (Value when unspecified) shall specify a default range for the dimension that the RTI is to use in overlap calculations if the dimension is an available dimension of an attribute or interaction and has been left unspecified when a federate creates a region that is subsequently used (either for update or for subscription) with the attribute or interaction. The default range is specified as a nonnegative integer subrange of [0, dimension upper bound). Ranges shall be specified either by two values separated by two periods or by a single value. All ranges are closed on the lower bound (indicated by a left bracket), and open on the upper bound (indicated by a right parenthesis). The use of a single value shall specify a "point" range with the specified point as the lower bound and specified point plus one (1) for the upper bound, so that no other valid value lies between these points. An entry of "Excluded" indicates that the dimension shall not be used in overlap calculations unless a specific range is provided in the region specification.

4.6.3 Inclusion criteria

Federations that require more data reduction than the class-based filtering provided by DM services should use the dimension table to document federation-wide agreements on the semantics and use of DDM dimensions. The type and normalization function information is guidance to federates on the consistent use of dimensions to maintain correct DDM semantics across the federation. Similarly, federates that normally operate in such federations should document the mechanisms that they use to take advantage of DDM in their SOM. If DDM services are not used, no entries are required is this table.

4.6.4 Example

Table 13 illustrates an example of the use of the dimension table. The first dimension, *SodaFlavor*, is an enumerated dimension, and its *linearEnumerated* normalization function (see Annex B) would map *Cola* to [0 .. 1), *Orange* to [1 .. 2), and *RootBeer* to [2 .. 3). The next two dimensions, *BarQuantity* and *WaiterId*, are integer-based dimensions and would be mapped at uniform intervals across the ranges [0 .. 25) and [0 .. 20), respectively. Notice that the default *Value When Unspecified* for *BarQuantity* is a point range that is equivalent to [0 .. 1), and also that a note has been attached to the full table.

Table 13—Dimension table example

Name	Datatype	Dimension upper bound	Normalization function	Value when unspecified
SodaFlavor	FlavorType	3	linearEnumerated (Flavor, [Cola, Orange, RootBeer])	[0 .. 3)
BarQuantity	DrinkCount	25	linear (NumberCups, 1, 25)	[0]
WaiterId	EmplId	20	linear (WaiterId, 1, 20)	Excluded
Note	*[Note5]			

4.7 Time representation table

4.7.1 Purpose/background

Simulations provide a means of exercising system models over time. During a federation execution, time may play two roles. Federates may associate themselves with points on the HLA time axis, and they may associate some of their activities, such as updating an instance attribute's value or sending an interaction, with points on the HLA time axis. These points on the HLA time axis are referred to as timestamps. As a federation execution progresses, federates may advance along the HLA time axis.

The specific strategy used to advance time in a simulation is driven by the simulation's purpose. For example, faster-than-real time, event-stepped simulations are frequently used in analysis applications, whereas real time, time-stepped simulations are often used in training applications. The strategy chosen for advancing time is an important consideration in the design of a simulation, because it affects both simulation performance and the ability of a simulation to interoperate with other simulations in a federation. For this reason, it is important to document how a simulation supports the advancement of time in a federate's SOM and relevant SOM modules.

IEEE Std 1516.1-2010 provides a variety of time management services to control the advancement of federates along the HLA time axis. These services allow multiple federates with differing internal time advancement strategies to interoperate in a meaningful way. For this reason, it is vital for all federation members to agree on how timestamps are represented across the federation, and document this agreement in the federation's FOM and relevant FOM modules.

During federation execution, timestamps are represented as instances of the time representation abstract datatype (TRADT). This abstract datatype is provided to the RTI when a federate joins a federation execution so that an RTI knows how to use the timestamps. This table allows the datatype and semantics of the abstract datatype for timestamps to be explained in FOMs, FOM modules, SOMs, and SOM modules.

It is also important to define the lookahead characteristics of both federates and federations. At the individual federate level, the specific means by which lookahead is supported may be a factor in assessing compatibility with other potential federates for a given federation application. For federations, it is important to document a common set of characteristics for how lookahead will be supported across the full federation. Thus, SOMs, SOM modules, FOMs and FOM modules all include representations of lookahead as well as timestamps. Representations of lookahead shall be drawn from nonnegative datatypes.

An abstract datatype for lookahead is supplied when a federate joins a federation execution. The lookahead entry in this table allows the datatype and semantics of this abstract datatype to be documented in FOMs, SOMs, FOM modules, and SOM modules.

Semantic differences exist between the way time is represented for the purpose of depicting timestamps versus calculating lookahead. When depicting timestamps, time can be considered to be an absolute value on a timeline (the HLA time axis), and thus, time comparisons can be drawn to determine if one timestamp is greater than another. Lookahead, in contrast, represents a duration of time, which can be added to timestamps but is generally not used for comparison purposes.

Two time representations, HLAinteger64Time and HLAfloat64Time, are required to be supported by the HLA and are described in detail in IEEE Std 1516.1-2010. These types shall be provided by all RTIs. Federations may choose to use these default datatypes, supply their own time representations, or not supply any time representation (e.g., NA).

4.7.2 Table format

The template that shall be used for recording time representation is illustrated in Table 14.

Table 14—Time representation table

Category	Datatype	Semantics
Timestamp	<type>	<semantics>
Lookahead	<type>	<semantics>

The first column (Category) presents the two time-related values that are to be specified in the table: timestamp and lookahead. *Timestamp* and *Lookahead* correspond to the only two rows in this table.

The second column (Datatype) shall identify the datatype of the time value. This datatype shall be chosen from the name of any simple, enumerated, array, fixed record, or variant record datatype described in 4.13. “NA” shall be entered in this column when timestamp or lookahead information is inappropriate to the federate or federation.

The third column (Semantics) shall expand and describe the use of the datatype for time values. “NA” shall be entered in this column when semantics information is unnecessary.

4.7.3 Inclusion criteria

All federations shall document their use of timestamp and lookahead via the time representation table in a FOM and relevant FOM modules. All federates shall document their representation of timestamps via the time representation table in a SOM and relevant SOM modules to the extent that it is used in the federate. For federates that support lookahead, the representation of lookahead shall be documented via the time representation table in a SOM and relevant SOM modules.

4.7.4 Example

Table 15 provides an example of how the OMT tables are used to describe the representation of time for the restaurant application.

Table 15—Time representation table example

Category	Datatype	Semantics
Timestamp	TimeType	Floating point value expressed in minutes
Lookahead	LAType	Floating point value expressed in minutes (non-negative)
Note	NA	

4.8 User-supplied tag table

4.8.1 Purpose/background

The HLA RTI provides a mechanism for federates to supply tags with certain HLA services. These tags can be used to provide additional coordination and control over these services. The user-supplied tag table provides a means of documenting federation agreements regarding the datatype to be used with these tags.

4.8.2 Table format

The template that shall be used for recording information on user-supplied tags is illustrated in Table 16.

Table 16—User-supplied tag table

Category	Datatype	Semantics
Update/reflect	<type>	<semantics>
Send/receive	<type>	<semantics>
Delete/remove	<type>	<semantics>
Divestiture request	<type>	<semantics>
Divestiture completion	<type>	<semantics>
Acquisition request	<type>	<semantics>
Request update	<type>	<semantics>

The first column (Category) presents the HLA service categories that are capable of accepting a user-supplied tag.

— *Update/Reflect*: Updating and the corresponding reflection of instance attribute values.

— *Send/Receive*: Sending and the corresponding reception of an interaction.

— *Delete/Remove*: Deletion and the corresponding removal of an object instance.

— *Divestiture Request:* Negotiations involved in divesting ownership of instance attributes.

— *Divestiture Completion*: Confirmation and completion of the negotiated divestiture of ownership of instance attributes.

— *Acquisition Request*: Negotiations involved in acquiring ownership of instance attributes.

— *Request Update*. Requesting update of instance attribute values and the corresponding RTI direction to provide instance attribute values.

These seven service categories correspond to the only seven rows in Table 16.

The second column (Datatype) shall identify the datatype of the user-supplied tag for those categories of service that the federate or federation designate as providing a user-supplied tag. This datatype shall be chosen from the name of any simple, enumerated, array, fixed record, or variant record datatype described in subclauses of 4.13. "NA" shall be entered in this column when this category of user-supplied tag is not used by the federate or federation.

The third column (Semantics) shall expand and describe the use of the datatype for this user-supplied tag. "NA" shall be entered in this column when semantics information is unnecessary.

4.8.3 Inclusion criteria

All federations that make use of user-supplied tags shall document the appropriate representations of these tags in the user-supplied tag table in a FOM and relevant FOM modules. All federates that can make use of user-supplied tags shall document their tag representations via the user-supplied tag table in a SOM and relevant SOM modules.

4.8.4 Example

Table 17 provides an example of how the OMT tables are used to describe the representation of user-supplied tags by the restaurant federate. In this example, no tags are specified for update/reflect, send/receive, divestiture completion, or request update. A simple text string is used for delete/remove that describes the reasons for the action. A priority entry is specified for ownership transfer negotiations (both divestiture requests and acquisition requests). This entry indicates the criticality of the transfer; a high value tells the federate to accept or divest ownership immediately.

Table 17—User-supplied tag table example

Category	Datatype	Semantics
Update/reflect	NA	NA
Send/receive	NA	NA
Delete/remove	HLAASCIIstring	Reason for deletion
Divestiture request	PriorityLevel	High value for immediate transfer
Divestiture completion	NA	NA
Acquisition request	PriorityLevel	High value for immediate transfer
Request update	NA	NA
Note	NA	

4.9 Synchronization table

4.9.1 Purpose/background

The HLA RTI provides a mechanism for federates to synchronize activities using a feature called Synchronization Points. The synchronization table provides the means for a federate to describe the synchronization points that it is capable of honoring and for a federation to document agreements regarding synchronization points to be used.

4.9.2 Table format

The template that shall be used for recording information to be used in synchronization points is illustrated in Table 18.

Table 18—Synchronization table

Label	Tag datatype	Capability	Semantics
<label>	<type>	<capability>	<semantics>
<label>	<type>	<capability>	<semantics>
<label>	<type>	<capability>	<semantics>

The first column (Label) shall contain a text string that defines the label associated with a synchronization point. Synchronization labels shall adhere to HLA naming conventions (see 3.3.1).

The second column (Tag datatype) shall identify the datatype of the user-supplied tag for those synchronization points that the federate or federation designate as providing a user-supplied tag. If used, this datatype shall be chosen from the name of any simple, enumerated, array, fixed record, or variant record datatype described in 4.13. If no user-supplied tag is provided, a value of “NA” should be used for the tag datatype.

The third column (Capability) shall indicate the level of interaction that a federate is capable of honoring. For FOMs and FOM modules, this column does not apply and shall contain “NA.” For SOMs and SOM modules, valid values for this column are as follows:

- *Register*: The federate is capable of invoking HLA services to register the synchronization point.
- *Achieve*: The federate is capable of invoking HLA services to indicate achieving the synchronization point.
- *RegisterAchieve*: The federate is capable of both registering and achieving the synchronization point.
- *NoSynch*: The federate is capable of neither registering nor achieving the synchronization point.

The fourth column (Semantics) shall expand and describe the use of the synchronization point.

4.9.3 Inclusion criteria

All federations that make use of synchronization points shall document the appropriate representations of these points in the synchronization table of a FOM and relevant FOM modules. All federates that make use of synchronization points shall document them via the synchronization table in a SOM and relevant SOM modules.

4.9.4 Example

Table 19 provides an example of how synchronization points might be described for a particular federate. In this example, no tags are specified for the first three synchronization points and a tag of type *TimeType* is included in the fourth. This federate is able to achieve all of the synchronization points and to register the fourth as well.

Table 19—Synchronization table example

Label	Tag datatype	Capability	Semantics
InitialPublish	NA	Achieve	Achieved when all classes are published and subscribed, and all initially present objects are registered
InitialUpdate	NA	Achieve	Achieved when instance attribute values for all initially present objects are updated
BeginTimeAdvance	NA	Achieve	Achieved when time management services are invoked
PauseExecution	TimeType	Register Achieve	Achieved when the time advance after the time in the user-supplied tag is attained; time advance requests should then cease
Note	NA		

4.10 Transportation type table

4.10.1 Purpose/background

The HLA RTI provides different mechanisms for the transportation of data (interactions and object instance attribute values) among federates. The transportation type table provides a means for a federate designer to describe the types of transportation that can be supported and for federation designers to document agreements regarding transportation of instance attributes and interactions.

Two transportation types, HLAreliable and HLAbestEffort, are required by the HLA and are described in detail in IEEE Std 1516.1-2010. These types shall be provided by all RTIs. Other transportation types may be provided by specific RTIs.

4.10.2 Table format

The template that shall be used for recording information about transportation types is illustrated in Table 20. Note that the transportation types required by the HLA to be provided by all RTIs are predefined in the table. Other entries may be added after these entries.

Table 20—Transportation type table

Name	Reliability	Semantics
HLAreliable	Reliable	Provide reliable delivery of data in the sense that TCP/IP delivers its data reliably
HLAbestEffort	Best Effort	Make an effort to deliver data in the sense that UDP provides best-effort delivery
<name>	<reliability>	<semantics>

The first column (Name) shall contain a text string that defines the name associated with a transportation type. Transportation type names shall adhere to HLA naming conventions (see 3.3.1).

The second column (Reliability) shall indicate whether reliable delivery is guaranteed or not for this transportation. The following entries are valid: “Reliable” and “Best Effort.”

The third column (Semantics) shall describe the transportation type.

4.10.3 Inclusion criteria

The two transportation types (HLAreliable and HLAbestEffort) that are required by the HLA shall be included in this table. Federations may choose to use one, both, or neither of these types depending on their needs. Other transportation types that are provided by specific RTIs shall be included in this table when used by a federate or federation.

4.10.4 Example

Table 21 provides an example of how the OMT tables are used to describe the transportation types. The two required transportation types are followed by a user-defined type that delivers best effort data with higher priority.

Table 21—Transportation type table example

Name	Reliability	Semantics
HLAreliable	Reliable	Provide reliable delivery of data in the sense that TCP/IP delivers its data reliably
HLAbestEffort	Best Effort	Make an effort to deliver data in the sense that UDP provides best-effort delivery
PriorityBestEffort	Best Effort	Deliver best effort data with higher priority than HLAbestEffort.
Note	NA	NA

4.11 Update rate table

4.11.1 Purpose/background

The HLA RTI provides a mechanism for reducing the update rate in a federation. The update rate may be reduced by the RTI for attribute updates using best effort transportation. For attribute updates using reliable or best effort transportation, the RTI also provides signaling that enables owning federates to adjust their update rates. The update rate table provides a means for aligning specific names with maximum update rates. The use of update rates is described in detail in IEEE Std 1516.1-2010.

4.11.2 Table format

The template that shall be used for recording information about update rates is illustrated in Table 22.

Table 22—Update Rate table

Name	Maximum update rate
<name>	<value>
<name>	<value>
<name>	<value>

The first column (Name) shall contain a text string that defines the name associated with the update rate. Update rate names shall adhere to HLA naming conventions (see 3.3.1).

The second column (Maximum update rate) shall specify the associated maximum update rate in Hz using a decimal number greater than zero.

4.11.3 Inclusion criteria

All federations that make use of smart update rates shall document the appropriate representations of them in the update rate table of a FOM and relevant FOM modules. All federates that make use of smart update rates shall document them in the update rate table of a SOM and relevant SOM modules.

4.11.4 Example

Table 23 provides an example of how the OMT tables are used to describe three different maximum update rates.

Table 23—Update Rate table example

Name	Maximum update rate
High	30.0
Medium	5.0
Low	0.2
Note	NA

4.12 Switches table

4.12.1 Purpose/background

The HLA RTI performs actions on behalf of federates. Some of these actions may be enabled or disabled based on the capabilities of the federates or desires of the federation designers. These actions include automatically soliciting updates of instance attribute values when an object is newly discovered (controlled by the Auto Provide switch*)*; and advising federates when certain events occur (controlled by the advisory switches). The switches table permits the specification of the initial setting of each switch, indicating whether the RTI should perform these actions. The functionality of these switches is described fully in IEEE Std 1516.1-2010.

Although the initial setting of each switch is specified in this table, the value of each switch, excluding Delay Subscription Evaluation, may be changed during execution. Some switches are controlled on a federation-wide basis. For such switches, a federate making a change to one of these switch settings affects how the RTI interacts with all federates in the execution. The federation-wide switches are Auto Provide and Automatic Resign Action. Other switches are controlled on a per-federate basis. In such cases, each federate in a federation has the same initial settings, as indicated in this table, but a change to a switch setting is specified for an individual federate and the change only affects how the RTI interacts with that federate. For example, the RTI initializes each federate to report service invocations or not, as indicated by the Service Reporting switch found in this table; a federate making a change to this switch setting on another federate affects whether the RTI reports service invocations for the specified federate. The per-federate switches are Attribute Relevance Advisory, Attribute Scope Advisory, Convey Producing Federate, Convey Region Designators, Exception Reporting, Interaction Relevance Advisory, Object Class Relevance Advisory, and Service Reporting. Finally, the Delay Subscription Evaluation switch is a federation-wide-static switch and it cannot be changed during execution. The setting found in this table controls how the RTI interacts with all federates for the duration of the federation execution.

4.12.2 Table format

The switch settings shall be provided in a simple two-column table. The structure that shall be used to describe these settings is provided in Table 24.

Table 24—Switches table

Switch	Setting
Auto Provide	<auto provide>
Convey Region Designator Sets	<convey region designator sets>
Convey Producing Federate	<convey producing federate>
Attribute Scope Advisory	<attribute scope advisory>
Attribute Relevance Advisory	<attribute relevance advisory>
Object Class Relevance Advisory	<object class relevance advisory>
Interaction Relevance Advisory	<interaction relevance advisory>
Service Reporting	<service reporting>
Exception Reporting	<exception reporting>
Delay Subscription Evaluation	<delay subscription evaluation>
Automatic Resign Action	<automatic resign action>

The first column (Switch) presents the name of the switch whose setting shall be provided in this table. The definitions of these switches are as follows:

— *Auto Provide*: Whether the RTI should automatically solicit updates from instance attribute owners when an object is discovered.

— *Convey Region Designator Sets*: Whether the RTI should provide the optional *Sent Region Set* argument with invocations of *Reflect Attribute Values* and *Receive Interaction*.

— *Convey Producing Federate*: Whether the RTI should provide the optional Producing Federate with invocations of Discover Object Instances, Remove Object Instances, Reflect Attribute Values and Receive Interaction.

— *Attribute Scope Advisory*: Whether the RTI should advise federates when attributes of an object instance come into or go out of scope.

— *Attribute Relevance Advisory*: Whether the RTI should advise federates about whether they should provide attribute value updates for the value of an attribute of an object instance; the RTI bases this advisory on whether the value of the instance attribute is required by other federates.

— *Object Class Relevance Advisory*: Whether the RTI should advise federates about whether they should register instances of an object class; the RTI bases this advisory on whether other federates have expressed an interest in attribute(s) of the object class.

— *Interaction Relevance Advisory*: Whether the RTI should advise federates about whether they should send interactions of an interaction class; the RTI bases this advisory on whether other federates have expressed an interest in the interaction class.

— *Service Reporting*: Whether the RTI should report service invocations using MOM.

— *Exception Reporting*: Whether the RTI should report exceptions using MOM.

— *Delay Subscription Evaluation*: Whether the RTI should filter messages as soon as possible to reflect the current known federate subscriptions, if disabled, or if enabled, defer subscription evaluation until only just before delivery to the federate.

— *Automatic Resign Action*: What resign action the RTI shall perform when it invokes the *Resign Federation Execution* service on behalf of a federate that was lost due to a fault. The value shall be one of the following: UnconditionallyDivestAttributes, DeleteObjects, CancelPendingOwnershipAcquisitions, DeleteObjectsThenDivest, CancelThenDeleteThenDivest, NoAction.

The second column (Setting) shall specify the setting for the switch. For SOMs and SOM modules, these entries are optional; "NA" shall be entered for all rows where no value is appropriate. For FOMs and FOM modules, entries shall be provided for all rows; valid values (with the exception of *Automatic Resign Action,* whose valid values are described above) are as follows:

— *Enabled*: The switch is enabled, and the RTI should perform the action when appropriate.

— *Disabled*: The switch is disabled, and the RTI should not perform the action.

4.12.3 Inclusion criteria

The switch settings specified in this table shall be included in all FOMs and relevant FOM modules to document agreements among federation developers as to the initial values of the switches at the beginning of federation execution. The switch settings may optionally be provided in SOMs and SOM modules to indicate switch settings that are desired by federate designers.

4.12.4 Example

Table 25 shows an example of the switches table for the Restaurant federate. In this example, the *Auto Provide* entry indicates that the federate developer does not desire the RTI to solicit updates of attributes of newly discovered object instances, the *Convey Region Designator Sets* entry indicates that the RTI should not provide the *Sent Region Set* argument, the *Convey Producing Federate* entry indicates that Producing Federate should not be provided, the *Service Reporting* entry indicates that the RTI should not report service invocations using the MOM, the *Exception Reporting* entry indicates that the RTI should not report exceptions using the MOM, and the *Delay Subscription Evaluation* entry indicates that the RTI should filter messages as soon as possible to reflect the current known federate subscriptions. However, the RTI should advise the federate regarding when attributes of object instances come into or go out of scope (*Attribute Scope Advisory*), whether the federate should register object instances based on subscriptions of other federates (*Object Class Relevance Advisory*), whether the federate should provide attribute value updates of object instances based on the interest of other federates (*Attribute Relevance Advisory*), and whether the federate should send interactions based on the interest of other federates (*Interaction Relevance Advisory*). Finally, the "CancelThenDeleteThenDivest" option is selected as the *Automatic Resign Action.*

Table 25—Switches table example

Switch	Setting
Auto Provide	Disabled
Convey Region Designator Sets	Disabled
Convey Producing Federate	Disabled
Attribute Scope Advisory	Enabled
Attribute Relevance Advisory	Enabled
Object Class Relevance Advisory	Enabled
Interaction Relevance Advisory	Enabled
Service Reporting	Disabled
Exception Reporting	Disabled
Delay Subscription Evaluation	Disabled
Automatic Resign Action	CancelThenDeleteThenDivest
Note	NA

4.13 Datatype tables

4.13.1 Purpose/background

Several of the OMT tables (attribute table, parameter table, dimension table, time representation table, user-supplied tag table, and synchronization table) provide columns for datatype specifications. This subclause describes additional tables that shall be used to specify the characteristics of these datatypes. A datatype used in these tables shall be specified in a row of one of the following tables: simple datatype table, enumerated datatype table, fixed record datatype table, array datatype table, or variant record datatype table. As described in 4.13.3 through 4.13.8, members of these datatype tables may contain members of other datatype tables in order to create arbitrarily complex datatypes.

NOTE—That the name of any datatype or data representation shall be unique among all datatypes and data representations.

4.13.2 Inclusion criteria

All federations shall describe and document the datatypes that are referenced in any OMT table that requires the specification of datatypes (attribute table, parameter table, dimension table, time representation table, user-supplied tag table, and synchronization table) in their FOM and relevant FOM modules. They shall also describe and document datatypes referenced in other datatype tables. Similarly, federates shall document the datatypes referenced in these tables in their SOM and relevant SOM modules.

4.13.3 Basic data representation table

Basic data representation is the underpinning of all OMT datatypes. A set of widely-used basic data representations have been pre-defined for use as the basis for the OMT datatypes and this list may be expanded. These basic data representations cannot be directly used as a named datatype in any OMT datatype table, but rather are the basis upon which named datatypes are built. Table 26 illustrates the format that shall be used for the basic representation of data. The table includes a predefined set of data representations (*HLAinteger16BE, HLAinteger32BE, HLAinteger64BE, HLAfloat32BE, HLAfloat64BE, HLAoctetPairBE, HLAinteger16LE, HLAinteger32LE, HLAinteger64LE, HLAfloat32LE, HLAfloat64LE, HLAoctetPairLE, HLAoctet*) that shall be present in all FOMs, SOMs, and MIMs; other entries may be added after these entries. Although these entries shall be present, there is no requirement that federates or federations use or support all of these representations.

Table 26—Basic data representation table

Name	Size in bits	Interpretation	Endian	Encoding
HLAinteger16BE	16	Integer in the range $[-2^{15}, 2^{15} - 1]$	Big	16-bit two's complement signed integer. The most significant bit contains the sign.
HLAinteger32BE	32	Integer in the range $[-2^{31}, 2^{31} - 1]$	Big	32-bit two's complement signed integer. The most significant bit contains the sign.
HLAinteger64BE	64	Integer in the range $[-2^{63}, 2^{63} - 1]$	Big	64-bit two's complement signed integer. The most significant bit contains the sign.
HLAfloat32BE	32	Single-precision floating point number	Big	32-bit IEEE normalized single-precision format. See IEEE Std 754-1985.
HLAfloat64BE	64	Double-precision floating point number	Big	64-bit IEEE normalized double-precision format. See IEEE Std 754-1985.
HLAoctetPairBE	16	16-bit value	Big	Assumed to be portable among hardware devices.
HLAinteger16LE	16	Integer in the range $[-2^{15}, 2^{15} - 1]$	Little	16-bit two's complement signed integer. The most significant bit contains the sign.

Table 26—Basic data representation table *(continued)*

Name	Size in bits	Interpretation	Endian	Encoding
HLAinteger32LE	32	Integer in the range $[-2^{31}, 2^{31}-1]$	Little	32-bit two's complement signed integer. The most significant bit contains the sign.
HLAinteger64LE	64	Integer in the range $[-2^{63}, 2^{63}-1]$	Little	64-bit two's complement signed integer. The most significant bit contains the sign.
HLAfloat32LE	32	Single-precision floating point number	Little	32-bit IEEE normalized single-precision format. See IEEE Std 754-1985.
HLAfloat64LE	64	Double-precision floating point number	Little	64-bit IEEE normalized double-precision format. See IEEE Std 754-1985.
HLAoctetPairLE	16	16-bit value	Little	Assumed to be portable among hardware devices.
HLAoctet	8	8-bit value	Big	Assumed to be portable among hardware devices.
<name>	<size>	<interpretation>	<endian>	<encoding>
<name>	<size>	<interpretation>	<endian>	<encoding>

The first column (Name) shall identify the name of the basic data representation. Basic data representation names shall adhere to HLA naming conventions (see 3.3.1).

The second column (Size in bits) shall define the size of the data representation in terms of the number of bits contained in the data representation. Entries in this column shall be consistent with the associated encoding descriptions.

The third column (Interpretation) shall describe how the data representation should be interpreted.

The fourth column (Endian) shall describe how multiple bytes within the representation are arranged. Valid values shall be as follows:

— *Big*: The most significant byte comes first.

— *Little*: The least significant byte comes first.

The fifth column (Encoding) shall describe, in detail, the encoding of the data representation (e.g., the bit ordering) as delivered to and received from the RTI.

An example of the use of the basic data representation table is provided in Table 27. This example includes the pre-defined set of basic data representations and follows with a user-defined entry (UnsignedShort).

4.13.4 Simple datatype table

The simple datatype table shall be used to describe simple, scalar data items. Table 28 illustrates the format that shall be used for the simple datatype table. The table includes five predefined simple datatypes (*HLAASCIIchar, HLAunicodeChar, HLAbyte, HLAinteger64time, HLAfloat64time*) that shall be present in all FOMs, SOMs, and MIMs; additional entries may be added after them. Although these required datatypes shall be present, no requirement exists that federates or federations use them.

The first column (Name) shall identify the name of the simple datatype. Simple datatype names shall adhere to HLA naming conventions (see 3.3.1).

The second column (Representation) shall identify the basic data representation of this datatype. It shall be the name of a row in the basic data representation table.

Table 27—Basic data representation table example

Name	Size in bits	Interpretation	Endian	Encoding
HLAinteger16BE	16	Integer in the range $[-2^{15}, 2^{15}-1]$	Big	16-bit two's complement signed integer. The most significant bit contains the sign.
HLAinteger32BE	32	Integer in the range $[-2^{31}, 2^{31}-1]$	Big	32-bit two's complement signed integer. The most significant bit contains the sign.
HLAinteger64BE	64	Integer in the range $[-2^{63}, 2^{63}-1]$	Big	64-bit two's complement signed integer. The most significant bit contains the sign.
HLAfloat32BE	32	Single-precision floating point number.	Big	32-bit IEEE normalized single-precision format. See IEEE Std 754-1985.
HLAfloat64BE	64	Double-precision floating point number	Big	64-bit IEEE normalized double-precision format. See IEEE Std 754-1985.
HLAoctetPairBE	16	16-bit value	Big	Assumed to be portable among hardware devices.
HLAinteger16LE	16	Integer in the range $[-2^{15}, 2^{15}-1]$	Little	16-bit two's complement signed integer. The most significant bit contains the sign.
HLAinteger32LE	32	Integer in the range $[-2^{31}, 2^{31}-1]$	Little	32-bit two's complement signed integer. The most significant bit contains the sign.
HLAinteger64LE	64	Integer in the range $[-2^{63}, 2^{63}-1]$	Little	64-bit two's complement signed integer. The most significant bit contains the sign.
HLAfloat32LE	32	Single-precision floating point number	Little	32-bit IEEE normalized single-precision format. See IEEE Std 754-1985.
HLAfloat64LE	64	Double-precision floating point number	Little	64-bit IEEE normalized double-precision format. See IEEE Std 754-1985.
HLAoctetPairLE	16	16-bit value	Little	Assumed to be portable among hardware devices.
HLAoctet	8	8-bit value	Big	Assumed to be portable among hardware devices.
UnsignedShort	16	Integer in the range $[0, 2^{16}-1]$	Big	16-bit unsigned integer.
Note	NA			

The third column (Units) shall identify the units of measure (e.g., m, km, kg) for the datatype whenever such units exist. Any units entered in this column also specify the units of the entries in the Resolution and Accuracy columns that follow it. "NA" shall be entered in this column for datatypes without units.

The fourth column (Resolution) shall describe the precision of measure for the datatype. In general, this entry specifies the smallest resolvable increment between different values that can be effectively discriminated. In some situations, such as when values are stored in floating point datatypes, the resolution might vary with the magnitude of the value. Hence, in cases like this, a better sense of the resolution may be conveyed by the datatype. "NA" shall be entered in this column for datatypes for which resolution information does not apply.

The fifth column (Accuracy) shall describe maximum deviation of the value from its intended value in the federate or federation. This is ordinarily expressed as a dimensioned value, but it may also be *perfect* for

Table 28—Simple datatype table

Name	Representation	Units	Resolution	Accuracy	Semantics
HLAASCIIchar	HLAoctet	NA	NA	NA	Standard ASCII character (see ANSI X3.4-1986)
HLAunicodeChar	HLAoctetPairBE	NA	NA	NA	Unicode UTF-16 character (see *The Unicode Standard*, Version 3.0)*
HLAbyte	HLAoctet	NA	NA	NA	Uninterpreted 8-bit byte
HLAinteger64Time	HLAinteger64BE	NA	1	0	Standardized integer HLA time type according to IEEE Std 1516.1-2010
HLAfloat64Time	HLAfloat64BE	NA	NA	NA	Standardized float HLA time type according to IEEE Std 1516.1-2010
<simple type>	<representation>	<units>	<resolution>	<accu-racy>	<semantics>
<simple type>	<representation>	<units>	<resolution>	<accu-racy>	<semantics>

*IETF RFC 2781 [B4] defines UTF-16, UTF-16BE and UTF-16LE. UTF-16 with a BOM should be treated as UTF-16BE. UTF-16 with a FOM or SOM can be UTF-16BE or UTF-16LE.

discrete values. “NA” shall be entered in this column for datatypes for which accuracy information does not apply.

The sixth column (Semantics) shall describe the meaning and use of the datatype. “NA” shall be entered in this column when semantics information is unnecessary.

An example of the use of the simple datatype table is provided in Table 29. These datatypes are used in examples elsewhere in this document. The examples in this table are aimed at a big-endian platform; for a little-endian platform substitute BE with LE. Note that *DrinkCount* is used for attributes and for dimension definition, and *EmplId* is used for dimension and array definition.

4.13.5 Enumerated datatype table

The enumerated datatype table shall be used to describe data elements that can take on a finite discrete set of possible values. Table 30 illustrates the format that shall be used for the enumerated datatype table. The table includes a single predefined datatype (*HLAboolean*) that shall be present in all FOMs, SOMs, and MIMs; additional entries may be added after it. Although this required datatype shall be present, no requirement exists that federates or federations use it.

The first column (Name) shall identify the name of the enumerated datatype. Enumerated datatype names shall adhere to HLA naming conventions (see 3.3.1).

The second column (Representation) shall identify the basic data representation that forms the basis of this datatype. It shall be the name of a row in the basic data representation table that specifies discrete values.

The third column (Enumerator) shall provide the names of all enumerators associated with this datatype. Enumerator names shall adhere to HLA naming conventions (see 3.3.1).

The fourth column (Values) shall provide values that correspond to each enumerator in the same row. The values presented in the enumerator shall adhere to the data representation specified in the Representation

Table 29—Simple datatype table example

Name	Representation	Units	Resolution	Accuracy	Semantics
HLAASCIIchar	HLAoctet	NA	NA	NA	Standard ASCII character (see ANSI X3.4-1986)
HLAunicodeChar	HLAoctetPairBE	NA	NA	NA	Unicode UTF-16 character (see *The Unicode Standard*, Version 3.0)
HLAbyte	HLAoctet	NA	NA	NA	Uninterpreted 8-bit byte
HLAinteger64time	HLAinteger64BE	NA	1	0	Standardized integer HLA time type according to IEEE Std 1516.1-2010
HLAfloat64time	HLAfloat64BE	NA	NA	NA	Standardized float HLA time type according to IEEE Std 1516.1-2010
TimeType	HLAfloat32BE	Minutes	0.01667	NA	Time representation
LAType	HLAfloat32BE	Minutes	0.01667	NA	Time interval (nonnegative)
DollarRate	HLAfloat32BE	$/hour	0.01	Perfect	Cost per hour
Years	HLAinteger32BE	Years	1	Perfect	Elapsed time in years
DrinkCount	UnsignedShort	Cups	1	Perfect	Measure of number of drinks
EmplId	HLAinteger32BE	NA	1	Perfect	Employee identifier
RateScale	HLAinteger32BE	NA	1	Perfect	Evaluation of staff where 1 = best
Note	NA				

column. If multiple values are associated with a single enumerator, commas shall be used to separate the values.

The fifth column (Semantics) shall describe the meaning and use of this datatype. “NA” shall be entered in this column when semantics information is unnecessary.

Table 30—Enumerated datatype table

Name	Representation	Enumerator	Values	Semantics
HLAboolean	HLAinteger32BE	HLAfalse	0	Standard Boolean type
		HLAtrue	1	
<enumerated type>	<representation>	<enumerator 1>	<value(s)>	<semantics>
		…	…	
		<enumerator n>	<value(s)>	
<enumerated type>	<representation>	<enumerator 1>	<value(s)>	<semantics>
		…	…	
		<enumerator m>	<value(s)>	

An example of the use of the enumerated datatype table is provided in Table 31.

Table 31—Enumerated datatype table example

Name	Representation	Enumerator	Values	Semantics
HLAboolean	HLAinteger32BE	HLAfalse	0	Standard Boolean type
		HLAtrue	1	
PriorityLevel	HLAinteger32BE	Low	0	General three-level priority indicator
		Medium	1	
		High	2	
WaiterTasks	HLAinteger32BE	TakingOrder	1	Possible activities of waiters
		Serving	2	
		Cleaning	3	
		CalculatingBill	4	
		Other	5	
FlavorType	HLAinteger32BE	Cola	101	Possible flavors of soda
		Orange	102	
		RootBeer	103	
		Cream	104	
ExperienceLevel	HLAinteger32BE	Trainee	0	Level of experience of waiters
		Apprentice	1	
		Journeyman	2	
		Senior	3	
		Temporary	4	
		Master	5	
Note	NA			

4.13.6 Array datatype table

The array datatype table shall describe indexed homogenous collections of datatypes; these constructs are also known as arrays or sequences. Table 32 illustrates the format that shall be used for the array datatype table. The table includes four predefined array datatypes (*HLAASCIIstring, HLAunicodeString, HLAopaqueData, HLAtoken*) that shall be present in all FOMs, SOMs, and MIMs; additional entries may be added after them. Although these required datatypes shall be present, no requirement exists that federates or federations use them.

The first column (Name) shall identify the name of the array datatype. Array datatype names shall adhere to HLA naming conventions (see 3.3.1).

The second column (Element Type) shall identify the datatype of an element of this datatype. Values in this column shall be names from other datatype tables (simple datatype table, enumerated datatype table, fixed record datatype table, array datatype table, or variant record datatype table). Note that higher dimensional arrays may be specified by using array datatypes in the element type column of this table.

The third column (Cardinality) shall contain the number of elements that are contained in the array datatype. Multidimensional arrays can be specified via a comma-separated list of values, each representing one dimension. If the number of elements in the array varies during the use of the datatype, a range of values may be provided; if used, this range shall take the form of upper and lower bound values, separated by two

Table 32—Array datatype table

Name	Element type	Cardinality	Encoding	Semantics
HLAASCIIstring	HLAASCIIchar	Dynamic	HLAvariableArray	ASCII string representation
HLAunicodeString	HLAunicodeChar	Dynamic	HLAvariableArray	Unicode string representation
HLAopaqueData	HLAbyte	Dynamic	HLAvariableArray	Uninterpreted sequence of bytes
HLAtoken	HLAbyte	0	Zero-length array; no contents	Token representation
<array type>	<type>	<cardinality>	<encoding>	<semantics>
<array type>	<type>	<cardinality>	<encoding>	<semantics>

periods and surrounded by brackets. Alternatively, the keyword “Dynamic” may be entered into this column for these types of arrays.

The fourth column (Encoding) shall describe, in detail, the encoding of the array datatype (e.g., the sequence of elements and the order of elements in multi-dimensional arrays) as delivered to and received from the RTI. Encoding of elements in an array is determined by the datatype specified in the Element type column. Encoding of the array is determined by the encoding specified in the Encoding column. Array data should be interpreted first through the array encoding scheme, and then through the encoding scheme of the element’s datatype.

For one-dimensional array datatypes that use one of the two predefined array encodings, the appropriate keyword “HLAfixedArray” or “HLAvariableArray” shall be entered in the Encoding column. The “HLAfixedArray” encoding is intended for arrays with fixed cardinality and consists of the encoding of each element in sequence. The “HLAvariableArray” encoding is intended for arrays with variable (including dynamic) cardinality, and consists of the number of elements encoded as an *HLAinteger32BE*, followed by the encoding of each element in sequence. Full details of these predefined encodings are given in 4.13.9.

The fifth column (Semantics) shall describe the meaning and use of this datatype. “NA” shall be entered in this column when semantics information is unnecessary.

An example of the use of the array datatype table is provided in Table 33. *Employees* is a fixed-length array type that holds the employee identifiers of everyone currently working; It uses the standard *HLAfixedArray* encoding. *AddressBook* is an array type that can be used to hold a collection of addresses; because it is encoded as a sparse array, addresses can be updated selectively without sending every element of the array. Because one may want to update addresses selectively without sending every element of the array, a special encoding is defined for this datatype.

4.13.7 Fixed record datatype table

The fixed record datatype table shall be used to describe heterogeneous collections of types; these constructs are also known as records or structures. Each entry in the fixed record datatype table may contain fields that are of other types, such as simple datatypes, fixed records, arrays, enumerations, or variant records. This allows users to build “structures of data structures” according to the needs of their federate or federation. Table 34 illustrates the format that shall be used for the fixed record datatype table.

The first column (Record name) shall identify the name of the fixed record datatype. Record names shall adhere to HLA naming conventions (see 3.3.1).

Table 33—Array datatype table example

Name	Element type	Cardinality	Encoding	Semantics
HLAASCIIString	HLAASCIIchar	Dynamic	HLAvariableArray	ASCII string representation
HLAunicodeString	HLAunicodeChar	Dynamic	HLAvariableArray	Unicode string representation
HLAopaqueData	HLAbyte	Dynamic	HLAvariableArray	Uninterpreted sequence of bytes
HLAtoken	HLAbyte	0	Zero-length array; no contents	Token representation
Employees	EmplId	10	HLAfixedArray	Identifiers of employees currently working
AddressBook	AddressType	Dynamic	An HLAinteger32BE followed by a set of index-value tuples. Each tuple consists of an HLAinteger32BE indicating the array index, followed by the element for that index. The initial HLAinteger32BE indicates the number of index-value pairs to follow, since all array elements need not be included.	Collection of all employee addresses
Note	NA			

Table 34—Fixed record datatype table

Record name	Field			Encoding	Semantics
	Name	Type	Semantics		
<record type>	<field 1>	<type 1>	<semantics>	<encoding>	<semantics>
	…	…	…		
	<field n>	<type n>	<semantics>		
<record type>	<field 1>	<type 1>	<semantics>	<encoding>	<semantics>
	…	…	…		
	<field m>	<type m>	<semantics>		

The second column (Field name) shall identify the name of a field in the fixed record datatype. Field names shall adhere to HLA naming conventions (see 3.3.1).

The third column (Field type) shall identify the datatype of the field. Values in this column shall be names from other datatype tables (simple datatype table, enumerated datatype table, fixed record datatype table, array datatype table, or variant record datatype table).

The fourth column (Field semantics) shall describe the meaning and use of this field. “NA” shall be entered in this column when semantics information is unnecessary.

The fifth column (Encoding) shall describe, in detail, the encoding of the fixed record datatype (i.e., the organization of fields) as delivered to and received from the RTI. Encoding of fields in a fixed record is determined by the datatype specified in the Field Type column. Encoding of the fixed record is determined by the encoding specified in the Encoding column. Record data should be interpreted first through the record encoding scheme, and then through the encoding scheme of the field's datatype.

For fixed record datatypes that use the predefined fixed record encoding, the keyword "HLAfixedRecord" shall be entered in the Encoding column (see 4.13.9). The "HLAfixedRecord" encoding consists of the encoding of each field in sequence, in the order in which they are declared. Full details of this predefined encoding are given in 4.13.9.

The sixth column (Semantics) shall describe the meaning and use of this datatype. "NA" shall be entered in this column when semantics information is unnecessary.

An example of the use of the fixed record datatype table is provided in Table 35. These datatypes are used in examples elsewhere in this document. *ServiceStat* combines three *HLAboolean* values into one record. *AddressType* creates a record of components of a mailing address. Both datatypes use the standard *HLAfixedRecord* encoding.

Table 35—Fixed record datatype table example

Record name	**Field**			**Encoding**	**Semantics**
	Name	**Type**	**Semantics**		
ServiceStat	EntreeOk	HLAboolean	Entree status	HLAfixedRecord	Check-off on whether the server performed properly on elements of the meal
	Veggy1Ok	HLAboolean	Vegetable 1 status		
	Veggy2Ok	HLAboolean	Vegetable 2 status		
AddressType	Name	HLAASCIIstring	Employee name	HLAfixedRecord	Mailing address
	Street	HLAASCIIstring	Street address		
	City	HLAASCIIstring	City name		
	State	HLAASCIIstring	State abbreviation		
	Zip	HLAASCIIstring	Postal code		
Note	NA				

4.13.8 Variant record datatype table

The variant record datatype table shall describe discriminated unions of types; these constructs are also known as variant or choice records. Table 36 illustrates the format that shall be used for the variant record datatype table.

Table 36—Variant record datatype table

Record name	Discriminant			Alternative			Encoding	Semantics
	Name	Type	Enumerator	Name	Type	Semantics		
<variant type>	<name>	<type>	<set 1>	<name 1>	<type 1>	<semantics>	<encoding>	<semantics>
			…	…	…	…		
			<set n>	<name n>	<type n>	<semantics>		
<variant type>	<name>	<type>	<set 1>	<name 1>	<type 1>	<semantics>	<encoding>	<semantics>
			…	…	…	…		
			<set m>	<name m>	<type m>	<semantics>		

The first column (Record name) shall identify the name of the variant record datatype. Record names shall adhere to HLA naming conventions (see 3.3.1).

The second column (Discriminant name) shall identify the name of the discriminant. Discriminant names shall adhere to HLA naming conventions (see 3.3.1).

The third column (Discriminant type) shall identify the datatype of the discriminant. Values in this column shall be names from the enumerated datatype table.

The fourth column (Discriminant enumerator) shall be a set of enumerators that determines the alternative. The enumerators shall be from the enumerated datatype specified in the Discriminant type column. The set shall consist of one or more individual enumerators or enumerator ranges separated by commas, or the special symbol "HLAother." An enumerator range shall be specified by two enumerators, separated by two periods, and enclosed in square brackets. The enumerator range shall denote all enumerators in the enumerated datatype whose definition in the enumerated datatype table occurs in between the specified enumerators (including the specified enumerators themselves). The symbol "HLAother" shall denote all enumerators in the enumerated datatype that are not explicitly included in any of the other Enumerator entries for the associated record name. The symbol "HLAother" shall only occur at most once for each record name.

The fifth column (Alternative name) shall define the identifier or name for the alternative. Alternative names shall adhere to the HLA naming conventions (see 3.3.1). If no alternative applies to the enumerator described in the Discriminant enumerator column, "NA" shall be entered in this column.

The sixth column (Alternative type) shall identify the datatype of the field. Values in this column shall be names from other datatype tables (simple datatype table, enumerated datatype table, fixed record datatype table, array datatype table, or variant record datatype table). If no alternative applies to the enumerator described in the Discriminant enumerator column, "NA" shall be entered in this column.

The seventh column (Alternative semantics) describe the meaning and use of this alternative. "NA" shall be entered in this column when semantics information is unnecessary.

The eighth column (Encoding) shall describe, in detail, the encoding of the variant record datatype (e.g., the location of the discriminant) as delivered to and received from the RTI. Encoding of fields in a variant record is determined by the datatype specified in the Alternative type column. Encoding of the variant record is determined by the encoding specified in the Encoding column. Record data should be interpreted first through the record encoding scheme and then through the encoding scheme of the field's datatype.

For variant record datatypes that use the predefined variant record encoding, the keyword "HLAvariantRecord" shall be entered in the Encoding column (see 4.13.9). The "HLAvariantRecord" encoding consists of the discriminant followed by the alternative associated with the value of the discriminant. Full details of this predefined encoding are given in 4.13.9.

The ninth column (Semantics) shall describe the meaning and use of this variant record datatype. "NA" shall be entered in this column when semantics information is unnecessary.

An example of the variant record table is provided in Table 37. In this example, the WaiterValue datatype associated with the Efficiency and Cheerfulness attributes of the Waiter class is characterized as a variant record. The discriminant is based on the level of experience accumulated by the employee. For new employees, the alternative *CoursePassed* might be represented as an enumeration indicating whether the employee passed training, until the required tasks are learned. For more experienced employees, the alternative *Rating* might support a more expanded scale (e.g., 1–10) to support management decisions on relative pay increases and possible bonuses. For all other waiter types, i.e., temporary staff (presumably only hired for special functions), the alternative is empty. The discriminant itself is represented as an enumerated datatype, as is illustrated in Table 31.

Table 37—Variant record datatype table example

Record Name	Discriminant			Alternative			Encoding	Semantics
	Name	Type	Enumerator	Name	Type	Semantics		
Waiter Value	Val Index	Experience Level	Trainee	Course Passed	HLAboolean	Ratings scale for employees under training	HLAvariantRecord	Datatype for waiter performance rating value
			[Apprentice .. Senior], Master	Rating	RateScale	Ratings scale for permanent employees		
			HLAother	NA	NA	All others		
Note	NA							

4.13.9 Predefined encodings for constructed datatypes

Each of the constructed datatypes (arrays, fixed records, and variant records) has a predefined encoding that describes how the components (i.e., elements of an array, or fields of a fixed or variant record) within the constructed datatype are laid out with respect to each other. These encodings are defined precisely in the appropriate subclauses (4.13.9.1 through 4.13.9.4). These predefined encoding definitions include a description of how each component in a constructed datatype shall be padded in order to ensure proper byte alignment of the component that follows it. The use of any of the predefined keywords *HLAfixedArray, HLAvariableArray, HLAfixedRecord,* or *HLAvariantRecord* to describe the encoding for a constructed datatype shall indicate adherence to these padding requirements. Object modelers may define and use alternative encoding schemes as needed.

In general, padding bytes shall be added as necessary within a constructed type so that each component is properly aligned. For example, a 32-bit float should be aligned on a 32-bit boundary, and a 64-bit float on a 64-bit boundary. Determining the number of padding bytes needed after a component in a constructed type depends on up to three factors, as follows:

— The offset in bytes of that component from the beginning of the constructed type

— The size in bytes of that component

— The “octet boundary value” of the following component

The octet boundary value for simple datatypes and enumerated datatypes is derived from the type’s entry in the basic data representation table. The octet boundary value is defined as the smallest value 2^n, where *n* is a non-negative integer, for which (8×2^n) is greater than or equal to the size of the datatype in bits. Octet boundary values of the predefined basic data representations are as follows:

Basic representation	Octet boundary value
HLAoctet	1
HLAoctetPairBE	2
HLAinteger16BE	2
HLAinteger32BE	4
HLAinteger64BE	8
HLAfloat32BE	4
HLAfloat64BE	8
HLAoctetPairLE	2
HLAinteger16LE	2
HLAinteger32LE	4
HLAinteger64LE	8
HLAfloat32LE	4
HLAfloat64LE	8

The octet boundary value for a constructed datatype is the maximum octet boundary value of all components within that constructed datatype. For example, a fixed record containing an *HLAboolean* field (based on *HLAinteger32BE),* a field based on *HLAoctet* and a field based on *HLAfloat64BE*, has an octet boundary value of 8. Any constructed datatype that contains this fixed record has an octet boundary value of at least 8, and it may be larger depending on the octet boundary value of its other components.

The following subclauses describe each of the four predefined constructed datatype encodings, including the number of padding bytes to be added after the components of each constructed datatype. If these predefined encodings are used with any datatype whose size in bits is not an integer multiple of 8, padding *bits* shall be added to the datatype to increase its size in bits to the next integer multiple of 8. Padding bits and bytes are always represented by 0s (zeros).

4.13.9.1 HLAfixedRecord

The HLAfixedRecord encoding shall consist of the encoding of each component in sequence, in the order in which they are declared. The first component in a fixed record shall start at offset 0 of the record.

Zero or more padding bytes shall be added to each component, except the last, to ensure that the next component in the record is properly aligned. The number of padding bytes after such a component *i* of a fixed record with this encoding is the smallest nonnegative value of P_i that satisfies Equation (1):

$$(\text{Offset}_i + \text{Size}_i + P_i) \bmod V_{i+1} = 0 \qquad (1)$$

where

$Offset_i$	is the offset of the component i of the record (in bytes)
$Size_i$	is the size of the component i of the record (in bytes)
V_{i+1}	is the octet boundary value of component (i + 1) of the record

Padding bytes shall not be added after the last component of the record. The size of a fixed record shall include any added padding bytes.

For example, consider a fixed record using the *HLAfixedRecord* encoding that consists of a component based on *HLAoctet,* an *HLAboolean* component, and a component based on *HLAfloat64BE*, in that order. The encoding calculation is as follows:

Field 1: $(Offset_1 + Size_1 + P_1) \bmod V_2 = 0$

$(0 + 1 + P_1) \bmod 4 = 0$

$(1 + P_1) \bmod 4 = 0$

$P_1 = 3$

Field 2: $(Offset_2 + Size_2 + P_2) \bmod V_3 = 0$

$(4 + 4 + P_2) \bmod 8 = 0$

$(8 + P_2) \bmod 8= 0$

$P_2 = 0$

Field 3: No padding bytes are added after this, the last, component.

Thus, 3 padding bytes are added after the *HLAoctet*-based component and no padding bytes after the *HLAboolean* component or the *HLAfloat64BE*-based component. The total size of this fixed record is 16 bytes. A graphical representation of this example, showing the contents of each of the 16 bytes with padding bytes represented as 0s, is as follows:

Byte															
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
HLAoctet	0	0	0	HLAboolean				HLAfloat64BE							

Note that if the fixed record consisted of the same three components, but in the reverse order, no padding bytes are added after any component, and the size of this fixed record is 13 bytes.

4.13.9.2 HLAvariantRecord

The HLAvariantRecord encoding shall consist of the discriminant followed by the alternative associated with the value of the discriminant. The discriminant shall be placed at offset 0 of the record.

If no alternative applies to the value contained in the discriminant, padding bytes shall be added (if required) after the discriminant to ensure that its size in bytes is equal to its octet boundary value.

If there is an alternative for the value contained in the discriminant, padding bytes shall be added (if required) after the discriminant to ensure that the alternative is properly aligned. The number of padding bytes after such a discriminant of a variant record with this encoding is the smallest nonnegative value of P that satisfies Equation (2):

$$(Size + P) \bmod V = 0 \qquad (2)$$

where

Size	is the size of the discriminant (in bytes).
V	is the maximum of the octet boundary values of the alternatives.

Padding bytes shall not be added after the alternative. The size of the variant record shall include any added padding bytes.

For example, consider a variant record with a discriminant based on the *HLAoctet* representation, one alternative consisting of a field based on the *HLAinteger32BE* representation, and a second alternative consisting of a fixed record containing two fields, both based on the *HLAoctet* representation. The encoding calculation is as follows:

$(Size + P) \bmod V = 0$
$(1 + P) \bmod 4 = 0$
$P = 3$

Thus, 3 bytes of padding are added after the discriminant.

In a case in which the discriminant contains a value that indicates that the *HLAinteger32BE*-based alternative is used, a graphical representation of the variant record is as follows:

Byte							
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
HLAoctet	0	0	0	HLAinteger32BE			

In a case in which the discriminant contains a value that indicates that the fixed record alternative is used, a graphical representation of the variant record is as follows:

Byte					
0	**1**	**2**	**3**	**4**	**5**
HLAoctet	0	0	0	HLAoctet	HLAoctet

4.13.9.3 HLAfixedArray

The HLAfixedArray encoding is intended for arrays with fixed cardinality and shall consist of the encoding of each element in sequence. The first element of the array shall be placed starting at offset 0 of the array.

The number of padding bytes after each element *i* of an array, except the last element, with this encoding is the smallest nonnegative value of P_i that satisfies Equation (3):

$$(Size_i + P_i) \bmod V = 0 \qquad (3)$$

where

$Size_i$	is the size of the *i*th element of the array (in bytes)
V	is the octet boundary value of the element type

Padding bytes shall not be added after the last element of the array. The size of the fixed array shall include any added padding bytes.

For example, consider an array consisting of a fixed number of elements, each of which is a fixed record encoded using *HLAfixedRecord* and consisting of a field based on the *HLAinteger32BE* representation, followed by a field based on the *HLAoctet* representation.

The size of each element in this array is the same, 5 bytes, as calculated following the rules for the *HLAfixedRecord* encoding. The octet boundary value of each element in this array is 4 bytes, the maximum of the octet boundary values of the two fields of the record. The calculations for the padding after each element, except the last, are as follows:

$(Size_i + P_i) \bmod V = 0$
$(5 + P_i) \bmod 4 = 0$
$P_i = 3$

Thus, 3 bytes of padding are added after each element of the array, except for the last element. A graphical representation of such an array, containing exactly two elements, is as follows:

Byte												
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
HLAinteger32BE				HLA octet	0	0	0	HLAinteger32BE				HLA octet

4.13.9.4 HLAvariableArray

The HLAvariableArray encoding is intended for arrays with variable (including dynamic) cardinality and shall consist of a number_of_elements component encoded as an *HLAinteger32BE*, followed by the encoding of each element in sequence. The number_of_elements component shall start at offset 0 of the array.

Padding bytes, if required, shall be added after the number_of_elements component to ensure that the first element in the array is properly aligned. The number of padding bytes here is the smallest nonnegative value of *P* that satisfies Equation (4).

$$(4 + P) \bmod V = 0 \tag{4}$$

where

V is the maximum of the octet boundary value of the element type, and the octet boundary value of HLAinteger32BE.

The number of padding bytes after each element of a variable array is calculated in the same way as for *HLAfixedArray.* The size of the variable array shall include any added padding bytes.

For example, consider an array consisting of a variable number of elements based on the *HLAfloat64BE* representation. The padding calculation for the number_of_elements component is as follows:

$(4 + P) \bmod V = 0$
$\mathrm{V} = \max(8,4) = 8$
$(4 + P) \bmod 8 = 0$
$P = 4$

Thus, 4 bytes of padding are added after the number of elements component. A graphical representation of such an array, containing only one element, is as follows:

Byte															
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
HLAinteger32BE				0	0	0	0	HLAfloat64BE							

As a second example, consider an array, called *VarArray*, consisting of a variable number of elements based on a 24-bit representation. The calculation for padding after the number_of_elements component is as follows:

$(4 + P) \bmod V = 0$
$V = \max(4,4) = 4$
$(4 + P) \bmod 4 = 0$
$P = 0$

Thus, no padding is added after the number_of_elements component. The amount of padding after each element, except the last, is calculated as follows (using the formula for *HLAfixedArray* encoding):

$(Size_i + P_i) \bmod V = 0$
$V = \max(4,4) = 4$
$(3 + P_i) \bmod 4 = 0$
$P_i = 1$

Thus, 1 byte of padding is added after each element of the array, except for the last. A graphical representation of such an array, consisting of two elements, is as follows:

Byte										
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
HLAinteger32BE				24-bit value			0	24-bit value		

As a final example, consider a variable length array of *VarArray* elements as described above. In particular, consider an instance that contains two arrays, the first of which contains one element and the second two elements. Here, the entire array of arrays is referred to as the outer array, and each element is referred to as an inner array. Padding after the number_of_elements component of the outer array is calculated as follows:

$(4 + P) \bmod V = 0$
$V = \max(1,4) = 4$
$(4 + P) \bmod 4 = 0$
$P = 0$

No padding is added after the number_of_elements component of the outer array. The padding after the first element of the outer array is calculated as follows (using the formula for *HLAfixedArray* encoding):

$(Size_1 + P_1) \bmod V = 0$
$V = \max(1,4) = 4$
$(7 + P_1) \bmod 4 = 0$
$P_1 = 1$

Thus, 1 byte of padding is added after the first element of the outer array, which is the first inner array. No padding is added after the second element of the outer array, which is the second inner array. A graphical representation of this is as follows:

Byte															
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
HLAinteger32BE				HLAinteger32BE				24-bit value			0	HLAinteger32BE			

16	**17**	**18**	**19**	**20**	**21**	**22**
24-bit value			0	24-bit value		

4.14 Notes table

4.14.1 Purpose/background

Any entry within any of the OMT tables may be annotated with additional descriptive information outside of the immediate table structure. This feature permits users to associate explanatory information with individual table entries to facilitate effective use of the data.

The mechanism for attaching one or more notes to an OMT table entry shall be to include a notes pointer in the appropriate table cell. In the tabular OMT format, this notes pointer shall consist of a uniquely identifying note label (or a series of comma-separated labels) preceded by an asterisk and enclosed by brackets. The notes themselves shall be associated with the note label and included in this table. For table cells with more than one information entry (e.g., a class name and a publish/subscribe designation), each entry may receive a separate note. Also, a single note may be referenced numerous times in OMT tables and that a single OMT table entry may reference numerous notes.

4.14.2 Table format

The template that shall be used for recording notes is illustrated in Table 38.

Table 38—Notes table

Label	Semantics
<label>	<semantics>
<label>	<semantics>
<label>	<semantics>

The first column (Label) shall contain a label for every notes pointer referenced in the object model. Labels shall adhere to HLA naming conventions (see 3.3.1).

The second column (Semantics) shall contain the explanatory text that constitutes the note. A Uniform Resource Locator (URL) may be provided as a pointer to the desired content.

4.14.3 Inclusion Criteria

FOMs and SOMs describing all federations and federates, and relevant FOM modules and SOM modules used to formulate such FOMs and SOMs may include notes wherever such annotation improves the clarity and understandability of the object model.

4.14.4 Example

Table 39 provides an example of the use of the notes feature. Table 5 and Table 9 contain references to these notes.

Table 39—Notes table example

Label	Semantics
Note1	http://www.seasonalfoods.com/lobster
Note2	For most general-purpose restaurant simulations, this is *Subscribe* only.
Note3	Merit raises are not provided according to any regular time interval; they are provided on a supervisor's recommendation based on evidence of exceptional effort and performance.
Note4	Years of service are a factor in any merit raise. This value is only changed on the anniversary of the employee's hire.
Note5	These dimensions were reused from the "Tommy's Place" SOM.

5. FOM/SOM lexicon

5.1 Purpose/background

If interoperability among simulations is to be achieved, it is necessary not only to specify the classes of data required by the templates in Clause 4, but also to achieve a common understanding of the semantics of this data. The FOM/SOM lexicon provides a means for federations to define all object classes, interaction classes, object class attributes, and interaction parameters in their FOMs and relevant FOM modules, and for individual federates to define these terms in their SOMs and relevant SOM modules.

5.2 Object class definition table

This subclause provides the format for describing HLA object classes. The template that shall be used for this information is provided in Table 40.

The first column (Class) shall contain the names of all object classes described in the FOM, FOM module, SOM, or SOM module. Object class names shall use dot notation as necessary to uniquely identify the object class being defined and may include the full parentage of the class.

The second column (Semantics) shall describe the semantics for the object class.

Table 40—Object class definition table

Class	Semantics
<class>	<semantics>
…	…
<class>	<semantics>

Table 41 provides an example of the use of the table. The object classes are taken from the examples in the object class structure table (see Table 5).

Table 41—Object class definition table example

Class	Semantics
Customer	Patron of the restaurant
Employee	A person working for the restaurant
Employee.Dishwasher	Cleaner of plates, pots, and utensils
Note	NA

5.3 Interaction class definition table

This subclause provides the format for describing HLA interactions. The structure that shall be used for this information is provided in Table 42.

Table 42—Interaction class definition table

Class	Semantics
<class>	<semantics>
…	…
<class>	<semantics>

The first column (Class) shall contain the name of each interaction class. Interaction class names shall use dot notation as necessary to uniquely identify the interaction class being defined and may include the full parentage of the class.

The second column (Semantics) shall describe the semantics of the interaction class.

Table 43 provides an example of the use of the table. The interaction classes are taken from the examples in the interaction class structure table (see Table 7).

5.4 Attribute definition table

This subclause provides the format for describing the attributes that characterize object classes. The structure that shall be used for this information is provided in Table 44. A separate row shall be completed for each attribute of an object class.

Table 43—Interaction class definition table example

Class	Semantics
CustomerTransaction	The base class of interactions between customers and employee
CustomerTransaction.FoodServed	Waiter has served food
CustomerTransaction. FoodServed.DrinkServed	Waiter has served a drink
Note	NA

Table 44—Attribute definition table

Class	Attribute	Semantics
<class>	<attribute>	<semantics>
…	…	…
<class>	<attribute>	<semantics>

The first column (Class) shall contain the name of the object class to which a given attribute belongs. Dot notation shall be used as necessary to uniquely identify the object class and may include the full parentage of the class.

The second column (Attribute) shall contain the name of the attribute.

The third column (Semantics) shall describe the characteristic of the object class that this attribute is designed to model.

Table 45 provides an example of the use of the table. The object classes and attributes are taken from the examples in the attribute table (see Table 9).

Table 45—Attribute definition table example

Class	Attribute	Semantics
Employee	PayRate	Amount the employee is paid per hour
	YearsOfService	Number of years the employee has worked for the restaurant
	HomeNumber	Employee home phone number
	HomeAddress	Employee home address
Note	NA	

5.5 Parameter definition table

This subclause provides the format for describing the parameters that are associated with interaction classes. The structure that shall be used for this information is provided in Table 46. A separate row shall be completed for each parameter of an interaction class.

The first column (Class) shall contain the name of the interaction class with which a given parameter is associated. Interaction class names shall use dot notation as necessary to uniquely identify the interaction class and may include the full parentage of the class.

Table 46—Parameter definition table

Class	Parameter	Semantics
<class>	<parameter>	<semantics>
…	…	…
<class>	<parameter>	<semantics>

The second column (Parameter) shall contain the name of the parameter.

The third column (Semantics) shall describe the specific information that this parameter is designed to convey.

Table 47 provides an example of the use of the table. The interaction classes and parameters are taken from the examples in the parameter table (see Table 11).

Table 47—Parameter definition table example

Class	Parameter	Semantics
CustomerTransaction.FoodServed. MainCourseServed	TemperatureOk	Whether the food was the correct temperature
	AccuracyOk	Whether the correct food was served
	TimelinessOk	Whether the food was served in a reasonable amount of time
Note	NA	

6. Conformance

The OMT tables provide a common representation of an object model. This clause defines the measures for assessment of object models and object model tools, based upon the normative clauses of this specification. The spectrum of OMT tools is broad, but conformance is a narrow measure of the object models accepted or produced by a tool. Other tool capabilities, from user interface visualization to code generation, are significant but not assessed by these conformance measures. Within the HLA community, verification capabilities conduct these assessments experimentally and publicly certify the results.

6.1 Conformance labels

Both object models and tools may be assessed for conformance to this standard. Only when certified results are publicly available shall the label "HLA Conforming" be used.

6.1.1 Object model conformance

An HLA conforming object model shall document all of the OMT components of an HLA object model, be it a FOM or relevant FOM modules for federations or a SOM or relevant SOM modules for federates. The HLA-required table content, commonly referred to as MOM data, shall be provided as specified in Clause 11 of IEEE Std 1516.1-2010, and shall be contained in a MIM. Each OMT table shall be provided, with empty content only as permitted in the table descriptions. All entries in a conforming object model shall comply with the conventions given in 3.3 of this standard.

6.1.2 Object model tool conformance

An HLA conforming object model tool shall accept any conforming object model and operate on it. The only required operations shall be parsing the XML product form of the object model and, for tools with XML output, expressing the model in valid XML. Additional operations provided by the tool are selected by the developer. Operations with user input shall identify inputs that would cause the resulting object model to no longer be conforming. An example would be a user entered name that was not a valid XML name or did not comply with HLA naming conventions. The user shall have the option to produce a non-conforming object model, based on this input. Future acceptance of the non-conforming object model by conforming object model tools is desirable but not required.

6.1.3 XML product conformance

A conforming object model expressed in XML by a conforming object model tool shall be processed without validation exception by any fully conforming XML processor, as defined in REC-XML-20060816, Fourth Edition of XML Schema (www.w3.org), when validated against the OMT XML Schema defined in Annex E.

6.2 Conformance verification

Tests for conformance may cover any normative part of this standard, and conforming products are expected to comply with all "shall" statements. The following subclauses define the minimum requirements that shall be assessed in any conformance test. Invalid entries detected during conformance testing shall be explicitly identified.

6.2.1 Object model identification table

The following tests shall be assessed for the object model identification table:

1) The Name shall conform to the name conventions of 3.3.
2) The Modification Date shall conform to the format given in Table 1.
3) The remaining fields in this table shall not be blank (although some values may be "NA").

6.2.2 Object class structure table

The following tests shall be assessed for the object class structure table:

1) The Name shall conform to the name conventions of 3.3, all object classes shall be uniquely identifiable in an HLA object model when concatenated (via dot notation) with the names of higher-level superclasses.
2) The <P/S> field for each Name shall contain P, S, PS, or N.

6.2.3 Interaction class structure table

The following tests shall be assessed for the interaction class structure table:

1) The Name shall conform to the name conventions of 3.3, all interaction classes shall be uniquely identifiable in an HLA object model when concatenated (via dot notation) with the names of higher-level superclasses.
2) The <P/S> field for each Name shall contain P, S, PS, or N.

6.2.4 Attribute table

The following tests shall be assessed for the attribute table:

1) The Object shall match an object class given in the object class structure table.
2) The Attribute shall conform to the name conventions of 3.3, attribute names shall not duplicate (overload) the names of attributes of this class or any higher-level superclass.
3) The Datatype shall match a datatype name given in the datatype table.
4) The Update field shall contain Static, Periodic, Conditional, or NA.
5) The <D/A> field shall contain D, A, DA, or N.
6) The <P/S> field shall contain P, S, PS, or N.
7) The Available Dimensions shall match a dimension name given in the dimension table or NA.
8) The Transportation shall match an transportation name given in the transportation table.
9) The Order field shall contain Receive or TimeStamp.
10) The remaining fields in this table shall not be blank.
11) The table shall contain, for the Object HLAobjectRoot, the Attribute HLAprivilegeToDeleteObject.

6.2.5 Parameter table

The following tests shall be assessed for the parameter table:

1) The Interaction shall match an interaction class given in the interaction class structure table.
2) The Parameter shall conform to the name conventions of 3.3, parameter names shall not duplicate (overload) the names of parameters of this class or any higher-level superclass.
3) The Datatype shall match a datatype name given in the datatype table.
4) The Available Dimensions shall match a dimension name given in the dimension table or NA.
5) The Transportation shall match an transportation name given in the transportation table.
6) The Order field shall contain Receive or TimeStamp.

6.2.6 Dimension table

The Dimension table may not be present; if present the following tests shall be assessed for the dimension table:

1) The Name shall conform to the name conventions of 3.3.
2) The Datatype shall match a datatype name given in the datatype table.
3) The Dimension Upper Bound shall be a positive integer.
4) The Value When Unspecified shall match be a nonnegative integer subrange of [0, Dimension Upper Bound).
5) The remaining fields in this table shall not be blank.

6.2.7 Time representation table

The following tests shall be assessed for the time representation table:

1) Representations of lookahead shall be drawn from non-negative datatypes.
2) The Datatype shall be chosen from the name of any simple, enumerated, array, fixed record, or variant record datatype, or defined to be NA.

6.2.8 User-supplied tag table

The following test shall be assessed for the user-supplied tag table:

1) The Datatype shall be chosen from the name of any simple, enumerated, array, fixed record, or variant record datatype, or defined to be NA.

6.2.9 Synchronization table

The following tests shall be assessed for the synchronization table:

1) Synchronization labels shall adhere to the HLA naming conventions of 3.3.
2) The Tag Datatype shall be chosen from the name of any simple, enumerated, array, fixed record, or variant record datatype, or defined to be NA.
3) The Capability field shall have a value of NA for FOMs and FOM modules.
4) The Capability field shall contain one of the values Register, Achieve, RegisterAchieve, or NoSynch for SOMs and SOM modules.

6.2.10 Transportation type table

The following tests shall be assessed for the transportation type table:

1) The Name shall adhere to the HLA naming conventions of 3.3.
2) The Reliability field shall contain Reliable or Best Effort.

6.2.11 Update rate table

The following tests shall be assessed for the update rate table:

1) Update rate names shall adhere to the HLA naming conventions of 3.3.
2) The Maximum Update Rate field shall specify the associated maximum update rate in Hz using a decimal number greater than zero.

6.2.12 Switches table

The following tests shall be assessed for the switches table:

1) The value of the Automatic Resign Action switch shall be one of the following: UnconditionallyDivestAttributes, DeleteObjects, CancelPendingOwnershipAcquisitions, DeleteObjectsThenDivest, CancelThenDeleteThenDivest, NoAction.
2) The Setting field shall contain NA for SOMs and SOM modules.
3) The Setting field shall contain either Enabled or Disabled for FOMs and FOM modules.

6.2.13 Basic data representation table

The following tests shall be assessed for the basic data representation table:

1) The Name shall adhere to the HLA naming conventions of 3.3.
2) The Endian field shall contain Big or Little.

6.2.14 Simple datatype table

The following tests shall be assessed for the simple datatype table:

1) The Name shall adhere to the HLA naming conventions of 3.3.
2) The Representation field shall reference a row in the Basic Data Representation table.

6.2.15 Enumerated datatype table

The following tests shall be assessed for the enumerated datatype table:

1) The Name shall adhere to the HLA naming conventions of 3.3.
2) The Representation field shall reference a row in the Basic Data Representation table that specifies discrete values.
3) Enumerator names shall adhere to the HLA naming conventions of 3.3.

6.2.16 Array datatype table

The following tests shall be assessed for the array datatype table:

1) The Name shall adhere to the HLA naming conventions of 3.3.
2) The Element Type field shall contain names from other datatype tables (simple datatype table, enumerated datatype table, fixed record datatype table, array datatype table, or variant record datatype table).
3) The Cardinality field shall contain either a single value, a comma-separated list of values, a range of values (surrounded by brackets and separated by two periods), or Dynamic.
4) The Encoding field shall contain HLAfixedArray or HLAvariableArray if the predefined encodings are used.

6.2.17 Fixed record datatype table

The following tests shall be assessed for the fixed record datatype table:

1) The Record Name shall adhere to the HLA naming conventions of 3.3.
2) Field Names shall adhere to the HLA naming conventions of 3.3.
3) The Field Type field shall contain names from other datatype tables (simple datatype table, enumerated datatype table, fixed record datatype table, array datatype table, or variant record datatype table).
4) The Encoding field shall contain HLAfixedRecord if the predefined encoding is used.

6.2.18 Variable record datatype table

The following tests shall be assessed for the variant record datatype table:

1) The Record Name shall adhere to the HLA naming conventions of 3.3.
2) Discriminant names shall adhere to the HLA naming conventions of 3.3.
3) The Discriminant Type field shall contain names from the enumerated datatype table.
4) The Discriminant Enumerator field shall contain one or more enumerators or enumerator ranges separated by commas, or the special symbol HLAother.
5) Enumerator ranges shall be specified by two enumerators, separated by two periods, and enclosed in square brackets.
6) Values of HLAother in the Discriminant Enumerator field shall occur at most once for each record name.
7) Alternative names shall adhere to the HLA naming conventions of 3.3.
8) The Alternative Type field shall contain names from other datatype tables (simple datatype table, enumerated datatype table, fixed record datatype table, array datatype table, or variant record datatype table).
9) The Encoding field shall contain HLAvariantRecord if the predefined encoding is used.

6.3 Interface specification services usage table

6.3.1 Purpose/background

HLA supports federates with many different interface requirements.This table allows federate and federation developers to document the set of HLA Interface Specification services used by federates or federations. SOMs and FOMs may optionally include the Interface Services Usage Table as it is useful for both types of HLA object models.

From the SOM perspective, this table is useful because the HLA federate compliance testing process requires the submission of a statement documenting which HLA Interface services a federate uses. This statement is first used in the Conformance Cross Check test to confirm that the federate's SOM is congruent with the set of HLA Interface services being used. Later, this statement is used to help determine the federate Compliance Test Sequence. The inclusion of this table in a SOM would support the HLA Compliance Testing requirement.

From the FOM perspective, this table is useful because this table allows the HLA Interface Specification services used by federations and federates to be documented in FOMs and reused during federation preparations. During a federation execution, a federate may need to exercise only a subset of its capabilities. Federates may associate themselves with different federations, and federations may change federates from time to time. Federation developers and managers can use this table in two different ways. First, the table may be used as a tool to document a common set of services that will be supported across the full federation. As such, this table would be in a readily available, predefined form that would replace the current procedure of developing an ad hoc federation agreement to document HLA Interface service usage. The second way to use the table is as a mechanism to review the specific services a federate uses and then assess whether or not a federate is likely to be compatible with other federates that are already members of a specific federation. Both ways provide valuable assistance to federation developers and managers in the areas of reuse and interoperability during HLA Federation Development and Execution Process (FEDEP) activities.

6.3.2 Table format

The template that shall be used for service utilization is illustrated in Table 48. The second column (Service) shall provide the names of HLA services. The first column shall provide the HLA Interface Specification clause number associated with each service. The third column (Usage) shall identify the usage of that service within the federate or federation. Entries 4.5, 4.6, 4.9, and 4.10 shall be "Yes" for FOMs because these four services are the minimum required to form a valid federation. Entries 4.9 and 4.10 shall be "Yes" for SOMs because these two services are the minimum required to form a valid federate.

Table 48—Interface specification services usage table

IEEE Std 1516.1-2010 clause	Service	Usage
4.2	Connect	<usage>
4.3	Disconnect	<usage>
4.4	Connection Lost †	<usage>
4.5	Create Federation Execution	<usage>
4.6	Destroy Federation Execution	<usage>
4.7	List Federation Executions	<usage>
4.8	Report Federation Executions †	<usage>
4.9	Join Federation Execution	<usage>
4.10	Resign Federation Execution	<usage>
4.11	Register Federation Synchronization Point	<usage>

Table 48—Interface specification services usage table *(continued)*

IEEE Std 1516.1-2010 clause	Service	Usage
4.12	Confirm Synchronization Point Registration †	<usage>
4.13	Announce Synchronization Point †	<usage>
4.14	Synchronization Point Achieved	<usage>
4.15	Federation Synchronized †	<usage>
4.16	Request Federation Save	<usage>
4.17	Initiate Federate Save †	<usage>
4.18	Federate Save Begun	<usage>
4.19	Federate Save Complete	<usage>
4.20	Federation Saved †	<usage>
4.21	Abort Federation Save	<usage>
4.22	Query Federation Save Status	<usage>
4.23	Federation Save Status Response †	<usage>
4.24	Request Federation Restore	<usage>
4.25	Confirm Federation Restoration Request †	<usage>
4.26	Federation Restore Begun †	<usage>
4.27	Initiate Federate Restore †	<usage>
4.28	Federate Restore Complete	<usage>
4.29	Federation Restored †	<usage>
4.30	Abort Federation Restore	<usage>
4.31	Query Federation Restore Status	<usage>
4.32	Federation Restore Status Response †	<usage>
5.2	Publish Object Class Attributes	<usage>
5.3	Unpublish Object Class Attributes	<usage>
5.4	Publish Interaction Class	<usage>
5.5	Unpublish Interaction Class	<usage>
5.6	Subscribe Object Class Attributes	<usage>
5.7	Unsubscribe Object Class Attributes	<usage>
5.8	Subscribe Interaction Class	<usage>
5.9	Unsubscribe Interaction Class	<usage>
5.10	Start Registration For Object Class †	<usage>
5.11	Stop Registration For Object Class †	<usage>
5.12	Turn Interactions On †	<usage>
5.13	Turn Interactions Off †	<usage>
6.2	Reserve Object Instance Name	<usage>
6.3	Object Instance Name Reserved †	<usage>
6.4	Release Object Instance Name	<usage>
6.5	Reserve Multiple Object Instance Names	<usage>
6.6	Multiple Object Instance Names Reserved †	<usage>
6.7	Release Multiple Object Instance Names	<usage>
6.8	Register Object Instance	<usage>
6.9	Discover Object instance †	<usage>
6.10	Update Attribute Values	<usage>
6.11	Reflect Attribute Values †	<usage>

Table 48—Interface specification services usage table *(continued)*

IEEE Std 1516.1-2010 clause	Service	Usage
6.12	Send Interaction	<usage>
6.13	Receive Interaction †	<usage>
6.14	Delete Object Instance	<usage>
6.15	Remove Object Instance †	<usage>
6.16	Local Delete Object Instance	<usage>
6.17	Attributes In Scope †	<usage>
6.18	Attributes Out Of Scope †	<usage>
6.19	Request Attribute Value Update	<usage>
6.20	Provide Attribute Value Update †	<usage>
6.21	Turn Updates On For Object Instance †	<usage>
6.22	Turn Updates Off For Object Instance †	<usage>
6.23	Request Attribute Transportation Type Change	<usage>
6.24	Confirm Attribute Transportation Type Change †	<usage>
6.25	Query Attribute Transportation Type	<usage>
6.26	Report Attribute Transportation Type †	<usage>
6.27	Request Interaction Transportation Type Change	<usage>
6.28	Confirm Interaction Transportation Type Change †	<usage>
6.29	Query Interaction Transportation Type	<usage>
6.30	Report Interaction Transportation Type †	<usage>
7.2	Unconditional Attribute Ownership Divestiture	<usage>
7.3	Negotiated Attribute Ownership Divestiture	<usage>
7.4	Request Attribute Ownership Assumption †	<usage>
7.5	Request Divestiture Confirmation †	<usage>
7.6	Confirm Divestiture	<usage>
7.7	Attribute Ownership Acquisition Notification †	<usage>
7.8	Attribute Ownership Acquisition	<usage>
7.9	Attribute Ownership Acquisition If Available	<usage>
7.10	Attribute Ownership Unavailable †	<usage>
7.11	Request Attribute Ownership Release †	<usage>
7.12	Attribute Ownership Release Denied	<usage>
7.13	Attribute Ownership Divestiture If Wanted	<usage>
7.14	Cancel Negotiated Attribute Ownership Divestiture	<usage>
7.15	Cancel Attribute Ownership Acquisition	<usage>
7.16	Confirm Attribute Ownership Acquisition Cancel-	<usage>
7.17	Query Attribute Ownership	<usage>
7.18	Inform Attribute Ownership †	<usage>
7.19	Is Attribute Owned by Federate	<usage>
8.2	Enable Time Regulation	<usage>
8.3	Time Regulation Enabled †	<usage>
8.4	Disable Time Regulation	<usage>
8.5	Enable Time Constrained	<usage>
8.6	Time Constrained Enabled †	<usage>
8.7	Disable Time Constrained	<usage>

Table 48—Interface specification services usage table *(continued)*

IEEE Std 1516.1-2010 clause	Service	Usage
8.8	Time Advance Request	<usage>
8.9	Time Advance Request Available	<usage>
8.10	Next Message Request	<usage>
8.11	Next Message Request Available	<usage>
8.12	Flush Queue Request	<usage>
8.13	Time Advance Grant †	<usage>
8.14	Enable Asynchronous Delivery	<usage>
8.15	Disable Asynchronous Delivery	<usage>
8.16	Query GALT	<usage>
8.17	Query Logical Time	<usage>
8.18	Query LITS	<usage>
8.19	Modify Lookahead	<usage>
8.20	Query Lookahead	<usage>
8.21	Retract	<usage>
8.22	Request Retraction †	<usage>
8.23	Change Attribute Order Type	<usage>
8.24	Change Interaction Order Type	<usage>
9.2	Create Region	<usage>
9.3	Commit Region Modifications	<usage>
9.4	Delete Region	<usage>
9.5	Register Object Instance With Regions	<usage>
9.6	Associate Regions For Updates	<usage>
9.7	Unassociate Regions For Updates	<usage>
9.8	Subscribe Object Class Attributes With Regions	<usage>
9.9	Unsubscribe Object Class Attributes With Regions	<usage>
9.10	Subscribe Interaction Class With Regions	<usage>
9.11	Unsubscribe Interaction Class With Regions	<usage>
9.12	Send Interaction With Regions	<usage>
9.13	Request Attribute Value Update With Regions	<usage>
10.2	Get Automatic Resign Directive	<usage>
10.3	Set Automatic Resign Directive	<usage>
10.4	Get Federate Handle	<usage>
10.5	Get Federate Name	<usage>
10.6	Get Object Class Handle	<usage>
10.7	Get Object Class Name	<usage>
10.8	Get Known Object Class Handle	<usage>
10.9	Get Object Instance Handle	<usage>
10.10	Get Object Instance Name	<usage>
10.11	Get Attribute Handle	<usage>
10.12	Get Attribute Name	<usage>
10.13	Get Update Rate Value	<usage>
10.14	Get Update Rate Value For Attribute	<usage>
10.15	Get Interaction Class Handle	<usage>

Table 48—Interface specification services usage table *(continued)*

IEEE Std 1516.1-2010 clause	Service	Usage
10.16	Get Interaction Class Name	<usage>
10.17	Get Parameter Handle	<usage>
10.18	Get Parameter Name	<usage>
10.19	Get Order Type	<usage>
10.20	Get Order Name	<usage>
10.21	Get Transportation Type Handle	<usage>
10.22	Get Transportation Type Name	<usage>
10.23	Get Available Dimensions for Class Attribute	<usage>
10.24	Get Available Dimensions for Interaction Class	<usage>
10.25	Get Dimension Handle	<usage>
10.26	Get Dimension Name	<usage>
10.27	Get Dimension Upper Bound	<usage>
10.28	Get Dimension Handle Set	<usage>
10.29	Get Range Bounds	<usage>
10.30	Set Range Bounds	<usage>
10.31	Normalize Federate Handle	<usage>
10.32	Normalize Service Group	<usage>
10.33	Enable Object Class Relevance Advisory Switch	<usage>
10.34	Disable Object Class Relevance Advisory Switch	<usage>
10.35	Enable Attribute Relevance Advisory Switch	<usage>
10.36	Disable Attribute Relevance Advisory Switch	<usage>
10.37	Enable Attribute Scope Advisory Switch	<usage>
10.38	Disable Attribute Scope Advisory Switch	<usage>
10.39	Enable Interaction Relevance Advisory Switch	<usage>
10.40	Disable Interaction Relevance Advisory Switch	<usage>
10.41	Evoke Callback	<usage>
10.42	Evoke Multiple Callbacks	<usage>
10.43	Enable Callbacks	<usage>
10.44	Disable Callbacks	<usage>

6.3.3 Inclusion criteria

This table is optional. All federations may document their use of HLA services via the Interface Specification Service Usage Table in a FOM and relevant FOM modules. All federates may document their use of HLA services via the Interface Specification Service Usage Table in a SOM and relevant SOM modules.

6.3.4 Example

Table 49 provides an example of how the OMT tables are used to describe the usage of HLA services. Included are an illustrative subset of the HLA services that shall be listed in the table.

Table 49—Interface specification services usage table example

Clause	Service	Usage
4.2	Connect	Yes
4.3	Disconnect	Yes
4.4	Connection Lost †	Yes
4.5	Create Federation Execution	No
4.6	Destroy Federation Execution	No
. . .	. . .	. . .

7. FOM module/SOM module merging rules

In the interest of reuse and efficiency, FOM modules and SOM modules may be combined to formulate FOMs and SOMs for federations and federates, respectively. In the case where such modules are to be merged in the creation of a FOM or SOM, the rules provided in this clause are intended to assist both the user and facilitate common tool functionality for the development of HLA object models.

The following rules shall apply when constructing FOMs or SOMs from FOM modules or SOM modules:

— All descriptions of object classes, interaction classes, attributes, parameters, dimensions, synchronization points, transportation types, update rates, datatypes, notes and FOM/SOM lexicon shall be merged.
— During the merge process, the FOM/SOM shall contain the references column where the "Composed from" field shall contain references to contributing FOM/SOM modules. All object model identification table sub-elements shall be left blank to be filled-in by the FOM/SOM developer after the merge process.
— The tables for time representation, switches, and user defined tags shall be provided in at least one FOM or SOM. The tables shall be identical to the corresponding tables in the FOM/SOM modules.
— The resulting FOM/SOM shall contain a service usage table if at least one of the contributing modules includes such a table. The value in the Usage column for each specific service shall be "Yes" if at least one contributing module has the value "Yes" in the Usage column for the same service. Otherwise, it shall be "No."
— The FOM shall contain a MIM, containing predefined constructs as described in the standard. A MIM that is extended by the user may be specified; otherwise the MIM as defined in the HLA standard shall be used. A SOM may also contain a MIM.

The principles used to combine FOM modules or SOM modules, described in Annex C, are intended to produce the results identified in Table 50. If the set of FOM modules or SOM modules, when combined, do not produce such results, then the FOM or SOM being assembled shall not be considered a compliant object model.

Table 50—FOM and SOM module merge results

OMT Component	Merge results	OMT component clause	Merge principle clause
Object model identification	Within the object model identification table, the Reference field of a resulting FOM or SOM shall contain “Composed from” references to each of the FOM module and/or SOM modules used. For all other object model identification table fields, it is the responsibility of the FOM/SOM developer to complete the required metadata elements.	4.1	C.1
Object classes	The resulting FOM or SOM shall contain the union of the object classes in all modules. If several modules contain regular definitions of a named object class, all regular definitions shall be identical with respect to name, sharing, semantics and the set of named attributes. Attribute properties shall be compared as described in a subsequent table row. Scaffolding object class definitions shall not be taken into account.	4.2	C.2
Interaction classes	The resulting FOM or SOM shall contain the union of the interaction classes in all modules. If several modules contain regular definitions of a named interaction class, all regular definitions shall be identical with respect to name, sharing, semantics, available dimensions, transportation type, order type and the set of named parameters. Datatypes, dimensions and transportation types shall be compared as described in a subsequent table row. Parameter properties shall be compared as described in a subsequent table row. Scaffolding interaction class definitions shall not be taken into account.	4.3	C.2
Attributes	The resulting FOM or SOM shall contain the union of the attributes in all modules. If several modules contain definitions of a named attribute for a particular class, all definitions shall be identical with respect to datatype, update type, update condition, sharing, ownership, available dimensions, transportation type, order type and semantics. Datatypes, dimensions, and transportation types shall be compared as described in a subsequent table row.	4.4	C.2
Parameters	The resulting FOM or SOM shall contain the union of the attributes in all modules. If several modules contain definitions of a named parameter for a particular class, all definitions shall be identical with respect to datatype and semantics.	4.5	C.2
Dimensions	The resulting FOM or SOM shall contain the union of the dimensions in all modules. If several modules contain definitions of a named dimension, all definitions shall be identical with respect to datatype, dimension upper bound, normalization function, value when unspecified and semantics. Datatypes shall be compared as described in a subsequent table row.	4.6	C.4

Table 50—FOM and SOM module merge results *(continued)*

OMT Component	Merge results	OMT component clause	Merge principle clause
Time representation	The resulting FOM or SOM shall contain the time representation table as provided in at least one FOM or SOM. If several modules contain the time representation table they shall be identical.	4.7	C.8
User supplied tags	The resulting FOM or SOM shall contain the user supplied tags table as provided in at least one FOM or SOM. If several modules contain the user supplied tags table they shall be identical.	4.8	C.8
Synchronization points	The resulting FOM or SOM shall contain the union of the synchronization points in all modules. If several modules contain definitions of a named synchronization points, all definitions shall be identical with respect to semantics.	4.9	C.5
Transportation type	The resulting FOM or SOM shall contain the union of the transportation types in all modules. If several modules contain definitions of a named transportation type, all definitions shall be identical with respect to reliability and semantics.	4.10	C.6
Update Rate	The resulting FOM or SOM shall contain the union of the update rates in all modules. If several modules contain definitions of a named update rate, all definitions shall be identical with respect to rate and semantics.	4.11	C.7
Switches	The resulting FOM or SOM shall contain the switches table as provided in at least one FOM or SOM. If several modules contain the switches table they shall be identical.	4.12	C.8
Datatypes	The resulting FOM or SOM shall contain the union of the datatypes in all modules. If several modules contain definitions of a named datatype, all definitions shall be identical with respect to all columns.	4.13	C.3.3
Notes	The resulting FOM or SOM shall contain the union of the notes in all modules. If several modules contain definitions of a named note, all definitions shall be identical with respect to semantics. If a repeated definition is provided and the original definition references a note, the repeated definition shall reference the same note using the same note identification.	4.14	C.7
Service usage	The resulting FOM/SOM shall contain a service usage table if at least one of the contributing modules includes such a table. The value in the Usage column for each specific service shall be "Yes" if at least one contributing module has the value "Yes" in the Usage column for the same service. Otherwise it shall be "No."	6.3	C.9

Annex A

(informative)

Table entry notation

The OMT table specifications use a subset of Backus Naur Form (BNF) [B5] to specify the types of entries that belong in particular table cells. In BNF, the types of terms to be substituted in the table are enclosed in angle brackets (e.g., <*class*>). Optional entries are enclosed in square brackets (e.g., *[<class> (<p/s>)]*). Any parentheses are terminal characters that should appear as shown. Thus, the basic entry in a cell of the object class structure table, designated by *<class> (<p/s>),* indicates a class name followed by a *Publish/ Subscribe* code in parentheses.

Annex B

(normative)

Common normalization functions

The following normalization functions are commonly used and may be referred to directly in dimension tables. The term DUB refers to the dimension upper bound specified for the dimension in the third column of Table 12. Ranges are indicated by two values separated by two periods and surrounded by brackets (indicating a closed range) or parentheses (indicating an open range).

B.1 Linear normalization function

Form linear (domain, dimensionLower, dimensionUpper)

Parameters *domain*: a nonenumerated value known to the federate using the dimension
dimensionLower: the lower bound on the value of *domain*
dimensionUpper: the upper bound on the value of *domain*

Function [(domain – dimensionLower) / (dimensionUpper – dimensionLower)] × (DUB – 1)

B.2 Linear enumerated normalization function

Form linearEnumerated (domain, mappedSet)

Parameters*domain*: an enumerated value known to the federate using the dimension
mappedSet: the enumerated values that *domain* can take (compact on the set of integers)

Function [positionInMappedSet(domain) /(mappedSetLength – 1)] × (DUB – 1)
where
positionInMappedSet(domain) is a function returning the position of *domain* in *mappedSet*, starting at zero
mappedSetLength is the number of elements in *mappedSet*

B.3 Enumerated set normalization function

Form enumeratedSet(domain, mappedSet)

Parameters *domain*: an enumerated value known to the federate using the dimension
mappedSet: the enumerated values that *domain* can take (not necessarily compact on the set of integers)

Function RTIset (domain)
where
RTIset (domain) is a function that maps *mappedSet* onto a range in the interval [0..DUB).

B.4 Logarithmic normalization function

Form	logarithmic(domain, domainLower, domainUpper)
Parameters	*domain*: a positive, nonenumerated value known to the federate using the dimension. *domainLower*: the lower bound on the value of *domain* (shall be positive) *domainUpper*: the upper bound on the value of *domain* (shall be greater than *domainLower*)
Function	[log(domain/domainLower) / log(domainUpper/domainLower)] × (DUB – 1)

B.5 Hyperbolic tangent normalization function

Form:	tanh(domain, domainCenter, domainSize)
Parameters:	*domain*: a nonenumerated value known to the federate using the dimension *domainCenter*: the value of *domain* around which the user desires the greatest precision *domainSize*: a generic measure of the distance around the *domainCenter* for which the user desires the relatively high precision
Function:	[{tanh([domain – domainCenter]/domainSize) + 1}/2] × (DUB – 1)

Annex C

(normative)

FOM module/SOM module merging principles

During the process of merging either FOM modules or SOM modules, as described in Clause 7, the resulting FOM or SOM may exist in an incomplete state. For federations, this state is defined by the FOM modules that have been provided thus far, and is referred to as the Current FOM. For federates, this state simply corresponds to the current developmental status of the SOM. All references to “FOM” or “SOM” in this clause can refer to either the current state of the object model or the finished state depending on the context.

When developing a FOM (or SOM), the first FOM module (or SOM module) selected provides the basis for the desired object model. Therefore, all the elements of the first FOM module (or SOM module) shall be incorporated into the FOM under development. The method for merging subsequent FOM modules (or SOM modules) should be performed using a top-down approach beginning with the object classes and interaction classes beginning at their root nodes. This is to confirm that all candidate object classes (and their attributes), interaction classes (and their parameters) and associated datatypes, dimensions, transportation, and notes that may be referenced by an object class or interaction class are properly considered for integration and are properly aligned when merged within the FOM (or SOM). A top-down approach requires that nodes representing a FOM module be walked down from the root node to the leaf nodes.

Following the parsing of object classes, interaction classes and their associated attributes, parameters, datatypes, dimensions, transportations, and notes, additional items that shall be considered for merging include the parsing of any defined synchronizations, updateRates, and remaining notes, which need to be compared and considered for integration into the FOM (or SOM).

C.1 Merging metadata

Each FOM module (or SOM module) used in the merging process shall be referenced in the resulting FOM’s (or SOM’s) object model identification table.Such references allow it to be known what FOM modules (or SOM modules) have been used in the construction of a FOM (or SOM), allowing their metadata to be further explored.Thus, for each module used in the creation of the FOM (or SOM), the value of the Reference Type field shall be “Composed_From,” and the value of the Reference Identification field shall contain the names of all FOM modules or SOM modules whose merger has resulted in the current object model.

With the exception of identifying such module references, all object model identification table sub-elements shall be left blank during the merge process. For all other object model identification table fields, it is the responsibility of the FOM/SOM developer to complete the required metadata elements. This information shall be completed at the conclusion of the merging process in order for the resulting FOM or SOM to be a compliant object model.

C.2 Merging classes

The classes (Object Classes/Interaction Classes) from FOM modules (or SOM modules) to be merged shall be compared against the current FOM (or SOM). Classes with the same name and ancestry (parent classes), whether or not they have different attributes/parameters, shall not be merged.

Any duplicates (equivalent and equivalent-like classes) already in the FOM (or SOM) are to be ignored. The following principles for checking for duplicacy of classes shall be applied to support such activity:

1) The class name of the FOM module (or SOM module) being inserted (including its ancestry) shall be compared to see if an equivalent class name (and ancestry) already exists in the FOM (or SOM).
2) If duplicacy is found, it is necessary to determine if the class that was to be added identically matched with the class that already exists.
3) Therefore, every sub-element of the class in the new module shall be compared to see if there is an identical sub-element in the class that already exists in the FOM (or SOM).
4) If some, but not all, sub-elements are equivalent then a warning shall be provided back to the user. This allows him/her to know that the class they desired to be inserted already exists but is not equivalent, and, therefore, cannot be inserted.
5) Comparison of sub-elements for an object class include: name, sharing (i.e., publish/subscribe capabilities), semantics, attributes, and notes. Children object classes are not compared. Within the attributes, what shall be compared include name, datatype, updateType, updateCondition, ownership, sharing, dimensions, transportation, order, semantics, and notes.
6) Sub-elements that are compared for an interaction class include: name, sharing, dimensions, transportation, order, semantics, parameters, and notes. Children interaction classes are not compared. Within the parameters, what shall be compared include name, datatype, semantics, and notes.
7) Within the datatypes of an attribute or parameter, the candidate datatype shall be compared to see if it already exists in the FOM (see C.3 on Datatype Merging.)

Any unique class names found in a FOM module (or SOM module) are considered candidates for being merged into the FOM (or SOM). The following principles shall be applied to support such activity:

— Check to see if an identical ancestry defined for the candidate class already exists in FOM (or SOM). This is necessary to determine compatibility of a candidate class, and to determine if any candidate class parents need to be also merged.
    a) If the parent lineage is found within the FOM (or SOM) and the new class and its subclass(es) do not introduce conflict with existing FOM (or SOM) elements (i.e., datatypes, synchronization points), then, under the equivalent parent in the FOM (or SOM), the class can be inserted.

       NOTE—Any subclasses associated to a candidate class shall also be identified as candidate classes. If a subclass has an equivalent class name as a class that exists within the FOM it may still be merged so long as the ancestry lineage is still unique.

    b) If no parent lineage is found within the FOM (or SOM) and the new class and its subclass(es) do not introduce conflict with existing FOM (or SOM) elements (i.e., datatypes, synchronization points), then the class and its parent classes can be merged into the FOM (or SOM).
— Within the datatypes of an attribute or parameter, the candidate datatype shall be compared to see if it already exists in the FOM (or SOM). (See C.3.)

C.3 Merging datatypes

Datatypes from FOM modules (or SOM modules) to be merged shall be compared against the first module used in the initial creation of the FOM (or SOM). Datatypes with the same name shall not be merged.

If the datatype does not exist in the FOM (or SOM), then the datatype from the FOM module (or SOM module) shall be identified as a candidate datatype to be inserted into the FOM (or SOM). Also, this includes the potential insertion of any other datatypes within the FOM module (or SOM module) that are used by the candidate datatype. Such insertion of these candidate datatypes shall only occur if the item (attributes, parameters, dimensions, synchronizationPoints) under consideration is inserted.

However, if the datatype does already exist in the FOM (or SOM), then the candidate datatype and all its sub-elements need to be compared for equivalency. Any duplicates (equivalent and equivalent-like datatypes) already in the FOM (or SOM) are to be ignored. The following principles for checking duplicacy of datatypes shall be applied to support such activity:

1) The candidate datatype name of a FOM module being inserted shall be compared to see if an equivalent datatype name already exists in the FOM (or SOM).
2) If duplicacy is found, it is necessary to determine if the datatype of the item (attribute, parameter, dimension, synchronizationPoint) that was attempted to be added identically matched with the datatype in the FOM (or SOM) that already exists.
3) If some but not all sub-elements of the datatype of the item (attribute, parameter, dimension, synchronizationPoint) that was attempted to be added are equivalent then a warning shall be provided back to the user. This allows him/her to know that the datatype or item (attribute, parameter, dimension, synchronizationPoint) they desired to be inserted already exists but is not equivalent, and, therefore, cannot be inserted. More specifically, that the item uses an inconsistent datatype and could not be added,
4) Any unique datatype names found in a FOM module (or SOM module) are candidates for being merged into the FOM (or SOM).

C.4 Merging dimensions

Dimensions from FOM modules (or SOM modules) to be merged shall be compared against the first module used in the initial creation of the FOM (or SOM). Dimensions with the same name shall not be merged. Any duplicates (equivalent and equivalent-like dimensions) already in the FOM (or SOM) are to be ignored. The following principles for checking duplicacy of dimensions shall be applied to support such activity:

1) The dimension name of FOM module (or SOM module) being inserted shall be compared to see if an equivalent dimension name already exists in the FOM (or SOM).
2) If duplicacy is found, it is necessary to determine if the dimension that was attempted to be added identically matched with the dimension that already exists.
3) If some but not all sub-elements of the dimension are equivalent then a warning shall be provided back to the user. This allows him/her to know that the dimension they desired to be inserted already exists but is not equivalent, and, therefore, cannot be inserted.
4) Sub-elements that are compared for a dimension include: name, datatype, upperBound, normalization, value, and notes.
5) Within the datatypes of a dimension, the candidate datatype shall be compared to see if it already exists in the FOM (SOM). (See C.3 on Datatype Merging.)

Any unique dimension names found in a FOM module (or SOM module) are candidates for being merged into the FOM (or SOM). The following principles shall be applied to support such activity:

— Check to see if the new dimension does not introduce conflict with any dimension sub-elements (i.e., datatypes) that already exist in the FOM (or SOM). This is necessary to determine if the dimension can be inserted under that parent in the FOM (or SOM).
— Within the datatypes of a dimension, the candidate datatype already exists in the FOM (or SOM). (See C.3 on datatype Merging.)

C.5 Merging synchronization points

Synchronization points from FOM modules (or SOM modules) to be merged shall be compared against the first module used in the initial creation of the FOM (or SOM). Synchronization points with the same label shall not be merged.

Any duplicates (equivalent and equivalent-like synchronizationPoints already in the FOM) are to be ignored. The following principles shall be applied to support such activity:

1) The synchronizationPoint label of FOM module being inserted shall be compared to see if an equivalent synchronizationPoint label already exists in the FOM (or SOM).
2) If duplicacy is found, it is necessary to determine if the synchronizationPoint that was attempted to be added identically matched with the synchronizationPoint that already exists.
3) If some but not all sub-elements of the synchronizationPoint are equivalent then a warning shall be provided back to the user. This allows him/her to know that the synchronizationPoint they desired to be inserted already exists but is not equivalent, and, therefore, cannot be inserted.
4) Sub-elements that are compared for a synchronizationPoint include: label, datatype, capability, semantics, and notes.
5) Within the datatypes of a synchronizationPoint, the candidate datatype shall be compared is to see if it already exists in the FOM (SOM). (See C.3 on Datatype Merging.)

Any unique synchronizationPoint labels found in a FOM module (or SOM module) are candidates for being merged into the FOM (or SOM). The following principles shall be applied to support such activity:

— Check to see if an identical ancestry defined for the candidate synchronizationPoint already exists in FOM (or SOM). This is necessary to determine compatibility of a candidate synchronizationPoint, and to determine if any candidate synchronizationPoint parents need to be also merged.
— If the parent lineage is found within the FOM (or SOM) and the new synchronizationPoint and its sub elements do not introduce conflict with existing FOM (or SOM) elements, then, under the equivalent parent in the FOM (or SOM), the synchronizationPoint can be inserted.
— If no parent lineage is found within the FOM (or SOM) and the new synchronizationPoint and its sub-elements do not introduce conflict with existing FOM (or SOM) elements, then the synchronizationPoint and its parent can be merged into the FOM (or SOM).
— Within the datatypes of a synchronizationPoint, the candidate datatype shall be compared is to see if it already exists in the FOM (or SOM). (See C.3 on Datatype Merging.)

C.6 Merging transportation types

Transportation types from FOM modules (or SOM modules) to be merged shall be compared against the first module used in the initial creation of the FOM (or SOM). Transportation types with the same name shall not be merged.

Any duplicates (equivalent and equivalent-like transportation types) already in the FOM (or SOM) are to be ignored. The following principles for checking duplicacy of transportation types shall be applied to support such activity:

1) The transportation type name of FOM module being inserted shall be compared to see if an equivalent to a transportation type name already exists in the FOM (or SOM).
2) If duplicacy is found, it is useful to determine if the transportation type that was attempted to be added identically matched with the transportation type that already exists.

3) If some but not all sub-elements of the transportation type are equivalent then a warning shall be provided back to the user. This allows him/her to know that the transportation type they desired to be inserted already exists but is not equivalent, and, therefore, cannot be inserted.
4) Sub-elements that are compared for transportation type include: name, semantics, and notes.

Any unique transportation type names are candidates for being merged into the FOM (or SOM).

— Check to see if an identical ancestry defined for the candidate transportation type already exists in FOM (i.e., candidate transportation type parents exist).
— If the new transportation type does not introduce conflict with existing FOM transportation type sub-elements, then the transportation type can be inserted under that parent in merged FOM (or SOM).

C.7 Merging update rates

Update rates from FOM modules (or SOM modules) to be merged shall be compared against the first module that was used in the initial creation of the FOM (or SOM). Update rates with the same name shall not be merged.

Any duplicates (equivalent and equivalent-like updateRates) already in the FOM (or SOM) are to be ignored. The following principles for checking duplicacy of update rates shall be applied to support such activity:

1) The update rate name of FOM module (or SOM module) being inserted shall be compared to see if an equivalent update rate name already exists in the FOM (or SOM).
2) If duplicacy is found, it is necessary to determine if the update rate that was attempted to be added identically matched with the update rate that already exists.
3) If some but not all sub-elements of the update rate are equivalent then a warning shall be provided back to the user. This allows him/her to know that the update rate they desired to be inserted already exists but is not equivalent, and, therefore, cannot be inserted.
4) Sub-elements that are compared for a update rate include: name, rate, semantics and notes.

Any unique update rate names found in a FOM module (or SOM module) are candidates for being merged into the FOM (or SOM). The following principles shall be applied to support such activity:

— Check to see if an identical ancestry defined for the candidate update rate already exists in FOM (or SOM). (i.e., candidate updateRate parents exist).
— If the new update rate does not introduce conflict update rate sub-elements contained within the existing FOM (or SOM), then the update rate can be inserted under that parent in merged FOM (or SOM).

C.8 Merging notes

Notes from FOM modules (or SOM modules) shall be parsed, and merged if they are referenced by any other table elements. The following principles for checking for duplicacy of notes shall be applied to support such activity:

1) Sub-elements that shall be compared for a Note include: label, semantics and notes.
2) Notes being selected for merge shall be reassigned unique IDs (labels) with in the Notes table. Additionally, any FOM module (or SOM module) element that references a note whose label has changed shall be updated prior to being merged within the FOM (or SOM).

C.9 Merging service usage tables

Service usage tables in FOM modules (or SOM modules) to be merged shall be compared against the service usage table in the current FOM/SOM if it exists. In case it exists a new value for the Usage column for each named row shall be calculated as a logical “or” of the Usage values of the FOM/SOM to be merged and the current FOM/SOM for the corresponding named rows. In case a service usage table does not exist in the current FOM/SOM, the service usage table of the FOM/SOM to be merged shall be inserted into the current FOM.

Annex D

(normative)

OMT data interchange format

The HLA OMT data interchange format (DIF) is a standard data exchange format used to store and transfer FOMs, SOMs, and HLA object model components (e.g., the HLA object modeling aspects of BOMs, as well as FOM modules, SOM modules, and MIMs) between object modeling tools and to initialize the HLA RTI. The DIF is built on a common metamodel that represents the information needed to represent and manage object models created using the HLA OMT standard.

D.1 Notation of the DIF

In order to ensure that the definition of the DIF is unambiguous, it has been formally defined in terms of XML, a formal notation used to describe document specifications. The standard for XML notation is described in REC-XML-20060816.

D.2 OMT DIF XML Schema declaration

XML calls for using an XML Schema or a Document Type Declaration (DTD) to specify the valid syntax, structure, and format for documents. The XML Schema that describes the OMT DIF is included in the paragraphs that follow.

The XML specification defines two levels of correctness for XML documents: “well-formed” and “valid.” A well-formed XML document is one that follows the syntax constraints of XML; for example, all element opening and closing tags are appropriately nested. A valid XML file is one that is well formed and complies with an XML Schema. That is, a valid XML document shall contain all elements and attributes defined in the XML Schema and only those elements and attributes. To support the exchange of object models among heterogeneous tools, all OMT DIF-compliant files shall be valid, and thus, well formed.

Because it is often necessary to exchange incomplete OMT documents using the OMT DIF (for federations under development, for example), many completeness and consistency constraints are not included in the OMT DIF XML Schema. Therefore, much of the individual information is specified as optional in the XML Schema. However, when complete, an OMT document, in addition to being a valid XML document, shall be consistent and complete with respect to the body of this document. In the OMT XML Schema, note labels and note references shall be prefixed with an underscore to ensure that the labeling conforms to the NCName datatype convention. Also, several instances exist in which the tabular representation differs from the XML representation, as follows:

— In the tabular representation of object models, object class and interaction class inheritance are represented by columns in the which the parent of a class was represented to its left and children to its right. In the XML depiction of object models, inheritance is represented by membership, in which children of a parent class are depicted as class elements contained within the parent class element.

— In the tabular representation, multiple entries in a field are represented by comma-separated lists. In XML, elements are separated by spaces. This situation exists in notes and attributes’ available dimensions.

— The formatting of notes in XML is different from that presented in the tabular representation in the body of this specification. In the tabular representation, a pointer to a note is indicated by an identifier enclosed in brackets in which the opening bracket is preceded with an asterisk. In XML, every XML attribute has a corresponding note attribute. This attribute shall receive the note pointers for the corresponding value attribute; the text of the note shall be recorded in the notes element.

The following file shall include the OMT DIF XML Schema. This file accompanies this specification.

http://standards.ieee.org/downloads/1516/1516.2-2010/IEEE1516-DIF-2010.xsd

Annex E

(normative)

OMT conformance XML Schema

The OMT Conformance XML Schema is more restrictive than the OMT DIF XML Schema. It enables automated conformance testing of OMT documents by using XML validation. Most but not all conformance tests, as defined in Clause 6, are expressed in the XML Schema.

Conformance tests not expressed in the OMT Conformance XML Schema are as follows:

1) Attribute names shall not duplicate (overload) the names of attributes of this class or any higher-level superclass.
2) Parameter names shall not duplicate (overload) the names of parameters of this class or any higher-level superclass

E.1 OMT conformance XML Schema declaration

The following file shall include the OMT DIF XML Schema. This file accompanies this specification.

http://standards.ieee.org/downloads/1516/1516.2-2010/IEEE1516-OMT-2010.xsd

Annex F

(informative)

OMT DIF SOM example

The HLA OMT DIF is described in Annex D. It describes the format and content of an object model in XML terms. The following is an object model document that encodes the samples describing the Restaurant SOM that are presented in this specification using the DIF. In XML terms, the document is well formed and valid according to the HLA OMT XML Schema specified in Annex D.

The following file shall include the Restaurant SOM module example. This file accompanies this specification.

http://standards.ieee.org/downloads/1516/1516.2-2010/RestaurantSOMmodule.xml

Annex G

(informative)

OMT DIF FOM example

The HLA OMT DIF is described in Annex D. It describes the format and content of an object model in XML terms. The following is an object model document that encodes a FOM module using the DIF. In XML terms, the document is well formed and valid according to the HLA OMT XML Schema specified in Annex D.

The following file shall include the Restaurant FOM example. This file accompanies this specification.

http://standards.ieee.org/downloads/1516/1516.2-2010/RestaurantFOMmodule.xml

Annex H

(informative)

Bibliography

[B1] IEEE Std 1516™-2010, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Framework and Rules.

[B2] IEEE Std 1516.3™-2003, IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP).

[B3] IEEE Std 1516.4™-2007, IEEE Recommended Practice for Verification, Validation, and Accreditation of a Federation—An Overlay to the High Level Architecture Federation Development and Execution Process.

[B4] IETF RFC 2781, “UTF-16, an encoding of ISO 10646,” February 2000.

[B5] Naur, P. *et al*., “Report on the Algorithmic Language ALGOL 60,” in *Communications of the ACM*, vol. 6, no. 1, pp. 1–17, Jan. 1963.

[B6] SISO-STD-003-2006, Base Object Model (BOM) Template Specification, Simulation Interoperability Standards Organization (SISO), March 2006.