**MUTHUR**

\_\_\_\_\_\_\_\_\_\_\_\_\_

**Requirements Document**

**Edition 0.1**

**16-Feb-11**

|  |  |  |
| --- | --- | --- |
| **DOCUMENT CHANGE HISTORY** | | |
| **Edition** | **Date** | **Description of Change** |
| 0.1 | 16-Feb-11 | First draft |
|  |  |  |

**Contents**

1 Introduction 4

1.1 Purpose of this Document 4

1.2 Related Documents 4

1.3 Terms and Definitions 4

1.4 Definition of MUTHUR 5

2 Requirements 6

2.1.1 Behavioral 6

2.1.2 Operational 10

2.1.3 Performance 11

3 Preliminary Database Table Schemas 12

Appendix A: Simulation Event Schema 12

# Introduction

## Purpose of this Document

This document lists and describes the operational, functional and technical requirements for the MUTHUR system.

MUTHUR provides synchronization, data sharing and ownership management, common model representation, time management and other related functions required to allow two or more simulations and/or applications to seamlessly interoperate and form a “federation” of systems that appear as a single, cooperating system. While it is intended and targeted at simulator or synthetic environments there is nothing in the design that limits participation in a federation to simulators.

While we intend to provide as complete a set of requirements as possible in the initial release of this document it is expected that requirements will evolve as we proceed. We fully expect this to be an evolving document that will change, as the phases of MUTHUR are completed and further information and knowledge of it is gathered and better understood.

## Related Documents

This section lists any documents that are referenced that are located in either the CSC Subversion repository or the CSC contracts database:

* Frasca/CSC sub-contract agreement

## Terms and Definitions

HLA – High Level Architecture, which is an architecture or framework developed by US government in conjunction with the DoD and NASA, upon which implementations for synchronizing and managing federations of simulation systems can be built.

Federation – logical grouping of systems, normally simulation systems that are participating in a coordinated and managed simulation or federation execution.

Federate – a participant in a federation execution.

Federation Object Model (FOM) – a description of the classes, attributes and attribute types that constitute the logical data model during the execution of a federation

Federation Execution Model (FEM) – data model describing the required participants or federates for the execution of a federation including the logical start and where appropriate the end time, required participants or federates, synchronization points and responsibilities of each federate such as time management and synchronization.

Synchronization Point – specific state during the federation execution to which participating federates will synchronize. Synchronization points are specified as part of the FEM and read by either MUTHUR proper or a federate responsible as the managing federate as so designated in the FEM.

## Definition of MUTHUR

The function of MUTHUR is to provide a centralized control system that allows multiple simulation systems to participate in a synchronized, coordinated fashion as a single federation of systems. Each participant in this federation is referred to as a federate. MUTHUR provides the federates the ability to:

* Define users and the roles as participants in simulations
* Establish a common data model for simulation runs or scenarios
* Control and manage participation in a federated simulation
* Synchronize participants to a single time clock and date in time across the federation
* Allow the creation and publication of data models
* Manage the instantiation of shared data objects
* Establish and manage ownership of these data objects between participants
* Allow participants to subscribe to instantiation, modifications and deletions of shared data
* Control the flow of the federated simulation through the life cycle of the federated simulation

MUTHUR is modeled after the ideas and capabilities from the HLA. The HLA is a specification or architecture that was borne out of the necessity in the DoD and NASA communities to coordinate and synchronize diverse and dispersed simulation systems and/or applications. The aim of the HLA was to provide a generic framework that would support as many of these types of systems as possible. Therefore, while it is very flexible it is also more complicated and generic then what is required of MUTHUR, at least in these initial phases.

MUTHUR is trying to achieve much of what the HLA encompasses but there is no organizational or contractual requirement that we are a fully HLA compliant implementation. While these requirements borrow many of the ideas and concepts from the HLA specifications it should not to be considered an full implementation of the HLA. Of course this may change in the future if and when customer, market or contractual requirements call for full HLA compliance. Therefore the design and implementation effort will take this into consideration and design accordingly in order to maintain the flexibility to move in this direction if required.

MUTHUR’s requirements can be split into the following high-level functional areas or services:

* Federation Management – defines the execution of a federation in terms of membership and existence. This may also encompass federation-wide saves, roll backs and check points for synchronization
* Subscription Management – allows federates to express interest in data instantiation, modification and/or deletion. The transformation of data is also specified via this service
* Object Management – manages the actual exchange of data between federates. Registration of new objects and their attributes and possible relationship to other existing objects can also be specified.
* Ownership Management – the ownership of and access to objects is handled through this service. Ownership may designated to be on at an object or attribute level. Ownership of different attributes within an object can also be shared across federates.
* Time Management – federates executing in their own threads or on different machines on the network require the proper ordering of events. The federation time is maintained as a logical value and not in terms of a real world time. Time management ensures that all federates advance in coordination with other federates. This service also ensures that a federate never receive events out of order.
* Data Distribution Management – controls the producer consumer relationship between federates and manages the routing definitions.

In addition to the functionality provided by the core of MUTHUR there is also a set of libraries or API that will reside at the client site. This library is referred to the Proxy and it is expected that each client will this set of libraries into their systems. The Proxy provides a simplified interface to the functionality of the MUTHUR. It encapsulates and removes the complexity of dealing directly with MUTHUR. It is largely based on an asynchronous model with results being returned via callbacks. The callbacks are defined in the underlying interface design specification for which the Proxy provides an implementation. The Proxy handles the communication, transformation, buffering and error handling between MUTHUR and the participating federate.

# Requirements

### Behavioral

The following sections briefly describe the functionality that is provided by MUTHUR to the federation. This functionality resides solely in the MUTHUR implementation with federates utilizing these functions via the Proxy.

#### User Management

Provides the functionality to establish users and roles and then associate users to their roles. While this can be accomplished via the Proxy it is expected that this capability will be utilized via a separate utility or function outside of the normal federates.

* Create a user and its attributes
* Modify a user’s attributes
* Delete a user
* Create a role
* Modify a role
* Add a role/user link
* Remove a role/user link
* Delete a role and all user links

#### Federation Management

* Register a FEM:
  + Unique identifier - GUID
  + FOM to be used
  + Required federates
  + Enumeration of supported synchronization points
  + Roles and responsibilities for each federate
  + Logical start time and where appropriate end time
* Read the list of registered FEMs
* Delete or remove registered FEM
* Update the required membership for a particular FEM
* Update the optional membership for a particular FEM
* Allow a federate to start a federation execution
* Allow a federate to join a federation execution providing a time out value which can be < 0 or wait indefinitely
* Delay start of federation execution until all participants have joined
* Abort federation execution if all participants have not joined before termination time period has elapsed
* Terminate or pause if any critical participants leave the federation execution
* Signal all federates once all required federates have joined the federation execution
* Signal all participating federates that the federation execution has started
* Signal all participating federates that they can publish classes and/or interactions
* Wait until all participants have completed publishing classes and/or interactions
* Signal all participating federates that federation execution has begun
* Signal all participating federates that the federation execution has paused and the time before resuming
* Signal all participating federates that the federation execution has stopped
* Wait until all participating federates signal they have stopped and cleaned up
* Signal all participating federates that the federation execution has terminated
* Allow a federate to signal a check point
* Signal all participating federates that a check point is in progress
* Signal all participating federates that a check point is complete
* Allow a federate to signal a roll back to a previous check point or back to a timestamp in the federation execution or the beginning of the federation execution if a checkpoint does not exist
* Signal all participating federates that a roll back is in progress
* Signal all participating federates that a roll back is complete

#### Subscription Management

Subscription management allows federates to express interest in the intent to publish classes, instantiation of objects, object modification, object deletion and publication of interactions. These classes, objects and interactions are limited to those that are defined within the FOM that is associated with the federation execution via the FEM. This is referred to as “the FOM”.

* Retrieve the FOM during a federation execution
* Subscribe to instantiation of any object type from the FOM
* Subscribe to modification of a specific instance of an object from the FOM
* Subscribe to deletion of a specific instance of an object from the FOM
* Register interest in modification of a specific attribute that exists on an instantiated object
* Unsubscribe to instantiation of any object type from the FOM
* Unsubscribe to modification of a specific instance of an object from the FOM
* Unsubscribe to deletion of a specific instance of an object from the FOM
* Unsubscribe to modification of a specific attribute that exists on an instantiated object
* Allow for the deployment of a specific transformation algorithm or transformer to be used for an object and it’s attributes
* Associate a transformer with a specific subscription
* Replace a subscription transformer with a previously associated transformer

#### Object Management

Manages the actual exchange of data between federates. Registration of new objects and their attributes and possible relationship to other existing objects can also be specified

* Publish the intent to publish objects instantiations, modifications and deletions for a particular class from a federate
* Publish a class definition and it’s attributes to all subscribers
* Publish new instance of a class to all subscribers
* Update an object and publish to all subscribers
* Update an object attribute and publish to all subscribers
* Publish the deletion of an object to all subscribers
* Publish an interaction to all subscribers

#### Data Distribution Management

Controls the producer consumer relationship between federates and manages the routing definitions. Federates are able to declare their intent to publish

* Add routing to support the subscriptions to any federate’s intent to publish instantiations, modifications and deletions for a specified class
* Add routing to support a subscription to object instantiation for a specified class
* Add routing to support a subscription to object modification for a specified class
* Add routing to support a subscription to object deletion for a specified class
* Add routing to support a subscription to all possible interactions
* Add routing to support a subscription to a specific interaction
* Remove routing to support a subscription to object instantiation for a specified class
* Remove routing to support a subscription to object modification for a specified class
* Remove routing to support a subscription to object deletion for a specified class
* Remove routing to support the subscriptions to any federate’s intent to publish instantiations, modifications and deletions for a specified class

#### Time Management

After the initial synchronization among the members of a federation has been achieved after the initial start of the federation execution it remains up to MUTHUR to synchronize the federation in terms of time.

While the initialization synchronization ensures that the federates will progress through the required steps in the proper order and in sequence with each other, they must also proceed throughout the execution stage in an ordered time sequence beginning at the same logical time.

The time management functions ensure that the federation proceeds forward in lock step providing the following functions:

* Set the initial logical time
* Receive requests to advance time
* Grant requests to all time regulated federates
* Notify federates that the end of federation execution condition has been achieved

#### Federate Requirements

Each participant or federate in the federation execution is required to abide a set of rules and follow a protocol. The protocol is provides for synchronization, for data exchange and for time management as the federation execution proceeds. It’s critical that all federates are synchronized to ensure that they progress to a set of known points or states. For the most parts all of these states or synchronization points are common to all federates. The coordination or synchronization of federates to these points is the responsibility of MUTHER. The fact that each federation follows the prescribed order of these synchronization points is the responsibility of each federate. The protocol or API is well understood and documented and each federate is required to incorporate that in the execution of it’s simulation or application.

A federate interacts with MUTHUR and thus communicates to the rest of the federation at the start and during execution through the client API. This API is supported in both the C++ and Java languages. The API contains both conventional synchronous calls as well as asynchronous call in the form of callbacks. It is through these callbacks that the other members of the executing federation as well as MUTHUR will communicate with a federate.

The following enumerates the steps and the expectations that the federation has for each participating federate:

* Join or start a federation execution
* Wait while join sync point has been achieved by federation
* Receive join sync point achieved notification from federation
* Publish all classes this federate intends to instantiate, update and delete during the execution as taken from the FOM which is associated with the FEM.
* Publish class publication has been completed by this federate
* Wait while ready to publish sync point has been achieved by federation
* Receive ready to publish sync point has been achieved by federation
* Instantiate and populate initial objects
* Publish ready to populate state has been completed by federate
* Wait while ready to populate sync point has been achieved by federation
* Receive ready to populate sync point has been achieved by federation
* Proceed with federation execution advancing time as one of the regulating federates or with the federation execution until receiving the ending condition
* Receive ending condition reached from federation and proceed with clean up and housekeeping chores
* Publish clean up completed by this federate
* Receive ending condition sync point achieved by federation

### Operational

This section describes the operational requirements for MUTHUR and for any participating federates in an executing federation.

* Network connectivity – each of the federates must be able to establish and maintain a connection to an instance of MUTHUR via standard network sockets. Connectivity should be done in as seamless a manner as possible without manual changes to configuration files or code to accommodate the specifics of the local network.
* Zero configuration – as with the network connectivity the level of effort required to accomplish a federation execution should be very low and at possible zero configuration is optimal.
* Robustness – while this not an operational or production but a simulation environment there is expectations that the federates participating in a federation execution be able to continue throughout the life of the execution without failure or system shutdown. Fatal errors and/or exceptions should be handled gracefully. Error conditions should be recorded and reported in order for system administrator to perform problem resolution.

### Performance

This section describes the performance expectations of the federation and of federates participating in a federation execution.

* Asynchronous responses

A federation is composed of one or more federates which will generally be distributed across a network. With this distributed, network configuration comes the issues of reliability and latency. With this a user cannot block on a call as in a tradition synchronous call since there’s no guarantee that the call will ever return. The Proxy design utilizes the design concept of call backs. The user calls one of the methods on Proxy with one of the parameters to the call being the implementation of an interface through which the results will be returned. The call returns almost immediately with the results being ultimately some time interval later in the appropriate method of the callback. There are several patterns that can be used to effectively deal with asynchronous calls, which are documented, in the design documents that follow this requirements document.

* Time slices

When a federate’s time is maintained or regulated it is said to be a regulated federate. The advancement of it’s internal time done by requests to advance and grants to advance given by MUTHUR. The duration of the federate’s time slice is determined by the federation designers. In order to ensure that the federation can execute smoothly and realistically it’s important that each federate is able to accomplish any time related tasks within this federation time slice.

* Initialization and housekeeping

Each federate is responsible for initialization at start up and clean up or housekeeping at the termination of the federation execution. As discussed previously the federation is synchronized through the initial steps before federation execution begins. Each federate is responsible for handling both the initialization and termination as quickly as possible with any lengthy tasks relegated to background processes and/or threads.

# Preliminary Database Table Schemas

1. Simulation Event Schema

This appendix describes the Federation Object Model that will be utilized by the federation execution consisting of a NexSim air traffic control (ATC) simulator and a Frasca International flight simulator or flight training device (FTD).

For each message there is a class in C and Java followed by the representation of the class in XML format. All messages have a “control section” containing the unique identifier for that message, message type, timestamp and source of the event. Beyond the control section is the specialized information for each message.

All message classes are derived from the ISimulationEvent interface which is extended by the abstract AbstractSimulationEvent class. Each class in the ISimulationEvent class hierarchy must support the input and output of the object in a well-formed XML document. Event objects are serialized as XML and then transmitted to their destination. Once received, they are reconstituted as an ISimulationEvent object using the XML received in the message.

<?xml version="1.0" encoding="UTF-8"?>

<simEventMessage xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'

simTime="2006-06-23T15:59:23" simSource="Frasca">

<flightEvent xsi:type="groundPositionEventType">

<IDData>

<acid>AAL101</acid>

<deptArptCode>JFK</deptArptCode>

<arrArptCode>ORD</arrArptCode>

<firstDeptTime>2006-06-23T15:50:00</firstDeptTime>

</IDData>

<positionData>

<latitudeDegrees>40.65</latitudeDegrees>

<longitudeDegrees>-73.72</longitudeDegrees>

<altitudeFt>11</altitudeFt>

<groundspeedKts>5</groundspeedKts>

<headingDegrees>90</headingDegrees>

</positionData>

</flightEvent>

</simEventMessage>

<?xml version="1.0" encoding="UTF-8"?>

<simEventMessage xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'

simTime="2006-06-23T15:59:23" simSource="Frasca">

<flightEvent xsi:type="airPositionEventType">

<IDData>

<acid>AAL102</acid>

<deptArptCode>JFK</deptArptCode>

<arrArptCode>ORD</arrArptCode>

<firstDeptTime>2006-06-23T15:55:00</firstDeptTime>

</IDData>

<positionData>

<latitudeDegrees>42.65</latitudeDegrees>

<longitudeDegrees>-76.72</longitudeDegrees>

<altitudeFt>30000</altitudeFt>

<groundspeedKts>400</groundspeedKts>

<headingDegrees>90</headingDegrees>

<airspeedKts>444</airspeedKts>

<pitchDegrees>0.5</pitchDegrees>

<rollDegrees>0</rollDegrees>

<yawDegrees>0</yawDegrees>

<sector>ZOB48</sector>

<center>ZOB</center>

</positionData>

</flightEvent>

</simEventMessage>

<flightEvent xsi:type="flightPlanEventType">

<IDData>

<acid>AAL103</acid>

<deptArptCode>IND</deptArptCode>

<arrArptCode>BGR</arrArptCode>

<firstDeptTime>2006-06-23T16:00:00</firstDeptTime>

</IDData>

<flightPlanData>

<source>Flight</source>

<aircraftType>B777</aircraftType>

<deptTime>2006-06-23T16:00:00</deptTime>

<arrTime>2006-06-23T18:00:00</arrTime>

<cruiseSpeedKts>400</cruiseSpeedKts>

<cruiseAltitudeFt>330</cruiseAltitudeFt>

<route>IND..ROD.J29.PLB.J595.BGR..BGR</route>

<deptCenter>ZID</deptCenter>

<arrCenter>ZBW</arrCenter>

<deptFix>DEFIX</deptFix>

<arrFix>ARFIX</arrFix>

<physicalClass>J</physicalClass>

<weightClass>H</weightClass>

<userClass>C</userClass>

<numAircraft>1</numAircraft>

<airborneEquipmentQualifier>R</airborneEquipmentQualifier>

</flightPlanData>

</flightEvent>

<flightEvent xsi:type="taxiOutEventType">

<IDData>

<acid>AAL104</acid>

<deptArptCode>IND</deptArptCode>

<arrArptCode>BGR</arrArptCode>

<firstDeptTime>2006-06-23T16:05:00</firstDeptTime>

</IDData>

<taxiOutData>

<taxiOutTime>2006-06-23T16:05:00</taxiOutTime>

</taxiOutData>

</flightEvent>

<flightEvent xsi:type="arrivalEventType">

<IDData>

<acid>AAL105</acid>

<deptArptCode>IND</deptArptCode>

<arrArptCode>BGR</arrArptCode>

<firstDeptTime>2006-06-23T16:10:00</firstDeptTime>

</IDData>

<arrivalData>

<actualArrTime>2006-06-23T18:05:00</actualArrTime>

</arrivalData>

</flightEvent>

<flightEvent xsi:type="cancelEventType">

<IDData>

<acid>AAL106</acid>

<deptArptCode>IND</deptArptCode>

<arrArptCode>BGR</arrArptCode>

<firstDeptTime>2006-06-23T16:15:00</firstDeptTime>

</IDData>

<cancelData>

<cancelTime>2006-06-23T16:15:00</cancelTime>

<cancellationFlag>RZ</cancellationFlag>

</cancelData>

</flightEvent>

<flightEvent xsi:type="controlEventType">

<IDData>

<acid>AAL106</acid>

<deptArptCode>IND</deptArptCode>

<arrArptCode>BGR</arrArptCode>

<firstDeptTime>2006-06-23T16:20:00</firstDeptTime>

</IDData>

<controlData>

<controlType>GDP</controlType>

<controlElement>BGR</controlElement>

<controlledDeptTime>2006-06-23T16:45:00</controlledDeptTime>

<controlledArrTime>2006-06-23T18:20:00</controlledArrTime>

</controlData>

</flightEvent>

<flightEvent xsi:type="summaryEventType">

<IDData>

<acid>AAL107</acid>

<deptArptCode>IND</deptArptCode>

<arrArptCode>BGR</arrArptCode>

<firstDeptTime>2006-06-23T16:25:00</firstDeptTime>

</IDData>

<positionData xsi:type="flightAirPositionType">

<latitudeDegrees>42.65</latitudeDegrees>

<longitudeDegrees>-76.72</longitudeDegrees>

<altitudeFt>30000</altitudeFt>

<groundspeedKts>400</groundspeedKts>

<headingDegrees>90</headingDegrees>

<airspeedKts>444</airspeedKts>

<pitchDegrees>0.5</pitchDegrees>

<rollDegrees>0</rollDegrees>

<yawDegrees>0</yawDegrees>

<sector>ZOB48</sector>

<center>ZOB</center>

</positionData>

<flightPlanData>

<source>Flight</source>

<aircraftType>B777</aircraftType>

<deptTime>2006-06-23T16:00:00</deptTime>

<arrTime>2006-06-23T18:00:00</arrTime>

<cruiseSpeedKts>400</cruiseSpeedKts>

<cruiseAltitudeFt>330</cruiseAltitudeFt>

<route>IND..ROD.J29.PLB.J595.BGR..BGR</route>

<deptCenter>ZID</deptCenter>

<arrCenter>ZBW</arrCenter>

<deptFix>DEFIX</deptFix>

<arrFix>ARFIX</arrFix>

<physicalClass>J</physicalClass>

<weightClass>H</weightClass>

<userClass>C</userClass>

<numAircraft>1</numAircraft>

<airborneEquipmentQualifier>R</airborneEquipmentQualifier>

</flightPlanData>

<taxiOutData>

<taxiOutTime>2006-06-23T16:05:00</taxiOutTime>

</taxiOutData>

<departureData>

<actualDeptTime>2006-06-23T16:45:00</actualDeptTime>

</departureData>

<controlData>

<controlType>GDP</controlType>

<controlElement>BGR</controlElement>

<controlledDeptTime>2006-06-23T16:45:00</controlledDeptTime>

<controlledArrTime>2006-06-23T18:20:00</controlledArrTime>

</controlData>

</flightEvent>

</simEventMessage>