4.0 Review of Continuous and Discrete Event Simulations

One of the key concepts in any simulation is the manner in which time is advanced. There is a basic distinction in this regard between simulations based on continuous models and those based on discrete-event models (Figure 4.1).

Continuous vs. Discrete Simulations

<table>
<thead>
<tr>
<th>Continuous</th>
<th>Discrete</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Continuously advances time and system state.</td>
<td>• System state changes only when events occur.</td>
</tr>
<tr>
<td>• Time advances in increments small enough to ensure accuracy.</td>
<td>• Time advances from event to event.</td>
</tr>
<tr>
<td>• State variables updated at each time step.</td>
<td>• State variables updated as each event occurs.</td>
</tr>
</tbody>
</table>

Figure 4.1 Continuous vs. Discrete Simulations

Continuous models assume continuously advancing time and a continuously changing system state. Computationally, this is represented by a time variable, which advances in increments small enough to accurately simulate the continuous behavior of the model. Updates to the variables representing the state of the system are typically computed for each step in time (Figure 4.2).
Discrete-event simulations are based on models that assume an unchanging system state until an event occurs that produces an instantaneous change in the state of the system. In these simulations, time advances from event to event in chronological time order.

4.1 HLA Classes, Objects, and Attributes

In the HLA, the different kinds of physical objects that are to be simulated are represented as classes (e.g., automobiles). Classes may be further divided into subclasses (e.g., 4-door and 2-door). Specific instances of a class are objects of that class (e.g., myCar and yourCar). Data that describes a simulated object are called the attributes of that object. The set of values of all of an object's attributes define the state of that object (Figure 4.3).
Simulation Objects

A program that simulates one or more classes of objects is known as a federate. Various federates (including multiple copies of some federates), running on one or more different machines, may join into a system simulation called a federation (Figure 4.4). The description of a federate's classes, objects, and attributes is given in the Simulation Object Model (SOM) for that federate. If this information is exchanged between federates, the information also appears in the federation's Federation Object Model (FOM) and FOM Document Data (FDD) files.
In an implementation of a federate using a specific object-oriented language such as C++, HLA classes may be represented as C++ classes, and HLA objects represented as instances of those C++ classes. HLA object attributes may be represented as those instance variables of an object that collectively define its state. Typically, routines for accessing and manipulating the object's attributes are included in the class. This is an obvious structure for implementing an HLA object, but it is not required by the standard. Also, care must be taken, because HLA is a language-independent standard, and the rules of any specific language for the behavior of classes and subclasses may not exactly match that defined for the HLA class structure.

An example that uses this implementation structure may be found in the HelloWorld federate. This federate defines a class Country, as the objects that are being simulated are specific countries in an environment consisting of a world of several countries. Numerous member functions are defined for class Country. Some are accessor-type functions such as Country::SetPopulation, which sets the population attribute for the Country to the specified value. Other functions in class Country actually provide simulation functionality, such as Country::Register, which registers its country with the federation.
4.2 Updates and Interactions

The state of the system is defined as a set of values, or attributes, for each object being simulated. Changes to the state of an object may be communicated to other federates in the federation by sending an update message to the federation using the RTI service Update Attribute Values (Figure 4.5). Other federates will receive the updated attribute values when the RTI calls the federate's Reflect Attribute Values† service. Recall that in addition to providing services to federates, the RTI requires a set of services from the federate that are referred to as "RTI initiated" and are denoted with a † (printer's dagger).

**HLA Messages -- Updates**

**Updates**
- Send/Receive new values of attributes at end of each time-step.
  - Send: Update Attribute Values
  - Recv: Reflect Attribute Values†
- Controlling unnecessary message traffic:
  - Update:
    - Enable/Disable Attribute Relevance Advisory Switch
    - Turn Updates On/Off for Object Instance†
  - Reflect:
    - Enable/Disable Attribute Scope Advisory Switch
    - Attribute In/Out of Scope†

Figure 4.5 HLA Messages -- Updates

There is a control mechanism to avoid sending unnecessary messages. This mechanism is enabled/disabled by the RTI service call: Enable/Disable Attribute Relevance Advisory Switch. The federate will be notified by the Turn Updates On/Off† RTI callback whether or not there are federates able to receive the updates. Sending updates should be suspended upon receiving the Turn Updates Off† callback from the RTI. Similarly, the federate that is interested in receiving updates will be notified whether or not there is any federate capable of sending the update if the Enable/Disable Attribute Scope Advisory Switch RTI service has been called. If no other federate is currently capable of sending the update, the Attributes Out of Scope† call will be made by the RTI. If an Attributes Out of Scope† call has been made by the RTI, it means that the federate's Reflect Attribute Values† routine will not be called by the RTI until the RTI first makes an Attributes In Scope† call.
If a simulation generates events, which cause an immediate change of state, it may send a message containing information about the event to other federates in the federation by using the RTI service: Send Interaction. Other federates may receive this message when the RTI calls the federate's Receive Interaction callback routine (Figure 4.6).

### HLA Messages -- Interactions

**Interactions**

- Send/Receive Parameters describing the event when an event occurs.
  - Send: Send Interaction
  - Recv: Receive Interaction
- Controlling unnecessary message traffic:
  - Enable / Disable Interaction Relevance Advisory Switch
  - Turn Interactions On/Off

![Figure 4.6 HLA Messages -- Interactions](image)

Interactions (events) have various data associated with them describing the characteristics (parameters) of the event. Events, like simulation objects, are divided into classes. An interaction class is declared which contains parameters quantifying the exact nature of the event. (For example, a traffic_light_color_change event might have a parameter that specifies the current color of the traffic light.) A specific event is considered a member of this interaction class, with specific values for its parameters. The mechanism for controlling unnecessary message traffic for interactions is enabled/disabled using the RTI services: Enable/Disable Interaction Relevance Advisory Switch. When the Interaction Relevance Advisory Switch is enabled for the joined federate (and only in such cases), the Turn Interactions On/Off services shall be invoked. The Turn Interactions On service shall notify the joined federate that the specified class of interactions is relevant because it or a super-class is actively subscribed to by at least one other federate in the federation execution. The Turn Interactions off service shall indicate to the joined federate that the specified class of interactions is not relevant because it or a super-class is not actively subscribed to by any other federate in the federation execution.
4.3 Publish and Subscribe Mechanism

It is possible that a federation has a large number of federates, simulating a wide variety of different physical entities. Not every federate is interested in receiving all of the messages generated by all other federates. For example, an automobile simulation may not have any need for data about an aircraft if there is no possible interaction between the aircraft and the automobile. The air traffic control simulation, however, may be very interested in the messages from this aircraft simulation. It is costly (in terms of simulation performance) to send unnecessary messages, and some kind of management is needed for the message delivery system of the federation (Figure 4.7).

The *publish* and *subscribe* mechanism is used by the HLA to manage the distribution of messages between the federates in a federation. (Both interactions and updates are considered messages.) Each federate defines to the federation what data are to be published (sent in a message), for each update or event using *Publish Object Class Attributes* and *Publish Interaction Class* RTI service calls. Each federate declares to the federation which updates and interactions (events) it is interested in receiving by subscribing to those messages using the *Subscribe Object Class Attributes* and *Subscribe Interaction Class* RTI service calls.

The “publish” calls are used to describe the data to be sent in messages to the RTI by a class of objects. As particular instances of these objects are created by a simulation, they are registered with the system by the RTI service call: *Register Object Instance*. If any federates have subscribed to objects of this class, they will receive a *Discover Object Instance* callback from the RTI that will inform them of the existence of the new object.

Notice that the subscribing federate does not need to know in advance exactly which, or how many, objects of a particular class will be of interest, only in which classes of objects it will be interested. (E.g., an automobile federate may be interested in all other automobiles, but not in aircraft. How many automobiles will be of interest depends on the number of automobiles that are introduced into the simulation.) Similarly, a federate that is publishing data does not need to know in advance anything about the federates that will be interested in subscribing to its data (Figure 4.7).
Object and Interaction Registration

<table>
<thead>
<tr>
<th>Objects</th>
<th>Interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Publish/Subscribe</td>
<td>• Publish/Subscribe</td>
</tr>
<tr>
<td>– Publish Object Class</td>
<td>– Publish Interaction Class</td>
</tr>
<tr>
<td>– Subscribe Object Class</td>
<td>– Subscribe Interaction Class</td>
</tr>
<tr>
<td>• Object Registration</td>
<td></td>
</tr>
<tr>
<td>– Register Object Instance†</td>
<td></td>
</tr>
<tr>
<td>– Discover Object Instance†</td>
<td></td>
</tr>
<tr>
<td>• Controlling Instance Registration:</td>
<td></td>
</tr>
<tr>
<td>– Enable/Disable Class Relevance Advisory Switch</td>
<td></td>
</tr>
<tr>
<td>– Start/Stop Registration for Object Class</td>
<td></td>
</tr>
</tbody>
</table>

If no other federate is interested in the data from objects of a particular class, e.g. simulated by federate “A”, then federate "A" could save simulation overhead if it stopped sending the data at all! There is a mechanism in the HLA for handling this situation. A federate may express an interest to the RTI in controlling the sending of unnecessary messages by using the RTI service Enable Attribute Relevance Advisory Switch for updates, or the RTI service Enable Interaction Relevance Advisory Switch for interactions. (Disable ... routines are available in both cases.) When any other federate subscribes to the data published by a federate (and who has issued the above enable commands to the RTI), the RTI will call the federate's Turn Updates On For Object Instance† or Turn Interactions On† callback routines. (Turn ... Off routines are also used in both cases.)

Also, there is no need to register an object of a certain class, if no federate exists that has subscribed to it. The RTI will call the federate routines Start/Stop Registration for Object Class† to control the object registration calls if the federate has previously called the RTI service Enable Object Class Relevance Advisory Switch. This advisory may be stopped by calling the RTI service: Disable Object Class Relevance Advisory Switch.
4.4 Time-Stamp Ordered and Receive Ordered Updates

Most simulations are written so that messages containing events or updates have an associated "time-stamp" and must be delivered to the receiving federate in the correct order with respect to this time-stamp. Messages that must be delivered in the correct time order are said to be delivered in *Time-Stamped Order* (TSO). A simulation that is publishing time-stamped data (TSO) may publish messages in any time order, and the RTI will guarantee to deliver the messages in correct time order (Figure 4.8).

Other types of messages, such as informational messages, may not have an associated time stamp, and are delivered upon arrival, without regard to the time when the message was sent. Messages delivered in the order in which they are received are said to be delivered in *Receive Order* (RO). An example might be an event that signals "The federation will shut down at 12:00." This message should be delivered as soon as it is available, asynchronously with respect to the current federate's simulated time.

### Message Order

Two types of Message Ordering

- **TSO (Time Stamped Order)**
  - Messages delivered to federate in order of time stamp
  - RTI guarantees that no messages will be received from past
- **RO (Receive Order)**
  - Messages delivered to federate in order received

**Figure 4.8 Message Order**

Time Management is concerned with mechanisms for controlling the advancement of time for each federate in consonance with the federation time axis. This advancement of time for each federate is controlled by the time management services so that federates will receive messages in the correct time order. No federate will receive messages from a time in the past (after it has already advanced time past the time stamp of the message).

4.5 Time-Regulating and Time-Constrained Federates
In order to guarantee the correct, time-ordered delivery of messages to a federate, the time for each federate must be coordinated with the time of other federates. Some federates may finish processing all known data before others; however, no federate could be allowed to suddenly advance its time to the next time step, until it has received all data up to that point in time from all other federates. Hence, there are limitations on the advancement of time by a federate that receives time-stamped messages. Those joined federates that receive TSO messages are called *time-constrained* (Figure 4.9).

**Regulating and Constrained Federates (1)**

- To receive TSO messages in Time Stamped Order, federate must declare itself *Time Constrained*.
- To send TSO messages, federate must declare itself to be *Time Regulating*.
- By default, federates are *neither* time constrained nor time regulating.
- To become time constrained, use RTI service *Enable Time Constrained*.
- To become time regulating, use RTI service *Enable Time Regulation*.

![Figure 4.9 Time Regulating and Time Constrained Federates](image)

In order for the time management services to properly control the advancement of time to time "T" for time-constrained federates, the RTI must be informed when all publishers of data are finished sending any messages up to time "T." The senders of time-stamped messages therefore *regulate* the advancement of time as simulation time cannot be advanced to time "T" until all publishers are finished sending data up to that time. Any federate that publishes time-stamped data is known as a *time-regulating* federate.

When a federate first joins the federation, it is in the default state of being neither *time-regulating* nor *time-constrained*. In this state, the federate will receive time-stamped messages in arbitrary order, as RO messages, until it becomes time-constrained. A federate will not correctly send time-stamped messages, as the messages will be sent without a time stamp as RO messages, until it becomes time-regulating. A joined federate that wishes to become time-regulating shall invoke the *Enable Time Regulation* service. The RTI shall subsequently make that joined federate time-regulating by invoking the *Time Regulation Enabled* service at that federate. Similarly, a joined...
federate may request to become time-constrained by invoking the *Enable Time Constrained* service. The RTI shall subsequently make that joined federate time-constrained by invoking the *Time Constrained Enabled* service at that federate. Federates may be both *time-constrained* and *time-regulating*, neither *time-constrained* nor *time-regulating*, or any combination of the two.

A joined federate shall cease to be time-regulating whenever it invokes the *Disable Time Regulation* service. Similarly, a joined federate shall cease to be time-constrained whenever it invokes the *Disable Time Constrained* service. In these cases, the RTI does not respond, and all future messages will be sent and/or received in receive order.

### 4.6 Lookahead, GALT, Logical Time, Effective Logical Time

Time in the system being modeled shall be represented in the federation as points along the HLA time axis. A joined federate can associate both itself and some of its activities with points on the HLA time axis. A joined federate's association with the HLA time axis is referred to as the *logical time* of that federate. An association of a joined federate's activities with the HLA time axis is denoted by assigning time stamps to the messages representing those activities (as discussed above). Time stamps and logical times are both represented by the same data type and so can be compared. However, they are referenced using different terms in order to help clarify whether a given time value applies to a joined federate itself or to a joined federate's activities.

Each joined federate, upon joining a federation execution, shall be assigned a logical time which initially shall be set to the initial time on the HLA time axis. If the federate is to receive time-stamped data from other federates, it must first become *time-constrained*. Time is "constrained" in the sense that a bound is placed on each time-constrained joined federate that limits how far it can advance its logical time. This is done in order to ensure that a time-constrained federate shall never receive a TSO message with a time-stamp less than its logical time. This bound on advancement of logical time is expressed in terms of a value called the *Greatest Available Logical Time* (GALT).

Each joined federate shall have a GALT that expresses the greatest logical time to which the RTI guarantees it can grant an advance without having to wait for other joined federates to advance. While GALT is only used by the RTI to bound the advancement of time-constrained joined federates, non-constrained joined federates also have a GALT. For a non-constrained joined federate, GALT expresses the bound that would apply to that federate if it were to become time-constrained.

Each joined federate shall also have another value called *Least Incoming Time stamp* (LITS) that builds on the idea of GALT. A joined federate's LITS shall express the smallest time stamp that the joined federate could (but not necessarily will) receive in
the future in a TSO message. A joined federate's LITS is calculated by the RTI and is based on that federate's GALT and any queued TSO messages that may later be received by the federate. If the joined federate's GALT is undefined and there are no queued TSO messages that the federate could receive, then the federate's LITS shall also be undefined. LITS is useful for joined federates wishing to know the time stamp of the next TSO message that they may have to process.

Remark: The IEEE 1516 version of the HLA specification introduces several changes in terminology. For example, the Greatest Available Logical Time (GALT) used to be called the Lower Bound Time Stamp (LBTS). The latter term is widely used in the literature. The Least Incoming Time Stamp (LITS) used to be called the Minimum Next Event Time (MNET) and the term "message" replaces the previously used term "event". Since we adhere to the latest available version of the HLA specifications (that is, to the IEEE 1516 document), we would like to specifically point out these changes in order to avoid any possible confusion and misunderstanding. (Figure 4.10).

Figure 4.10 Time-Constrained Federates

A joined federate's GALT is calculated by the RTI and is based on factors such as the logical time, lookahead, and requests to advance the logical time of time regulating joined federates. What is a "lookahead"? Each time-regulating joined federate, when it is becoming time regulating, shall provide a special non-negative value, called its lookahead, that establishes the lowest value of time stamps that can be sent in its TSO messages. More specifically, a time-regulating joined federate shall not send a TSO message that contains a time stamp less than its current logical time plus its current
lookahead. Once established, changes to a joined federate's lookahead may only be requested using the \textit{Modify Lookahead} service.

In order for a joined federate to advance its logical time, it shall request an advance explicitly and only by invoking one of the following services:

- \textit{Time Advance Request} (TAR)
- \textit{Time Advance Request Available} (TARA)
- \textit{Next Message Request} (NMR)
- \textit{Next Message Request Available} (NMRA)
- \textit{Flush Queue Request} (FQR)

The advance shall not occur until the RTI issues a grant through the \textit{Time Advance Grant} service. In general, at any instant during a federation execution, different joined federates may be at different logical times.

A time-regulating joined federate with a lookahead of \textit{zero} is subject to an additional restriction (compared to those with a \textit{positive} value of a lookahead). If such a joined federate has advanced its logical time by use of TAR or NMR, then it shall not send TSO messages that contain time stamps less than or equal to its logical time (plus its lookahead, which is zero), rather than the usual "less-than" restriction. Subsequent use of a time advancement service that moves the joined federate's logical time \textit{forward} lifts this additional restriction (and replaces it with the restriction of that subsequent advancement service) but if the subsequent advancement service does not move forward the federate's logical time, this additional restriction remains in place. For example, if a zero lookahead joined federate were to invoke \textit{Time Advance Request} (t1) and to follow this with an invocation of \textit{Time Advance Request Available} (t1), that joined federate would still have the additional restriction: after the \textit{Time Advance Request Available} is granted, the federate still may not send any TSO messages with a time stamp less than or equal to t1 (the \textit{Time Advance Request} restriction for zero lookahead federates) since the \textit{Time Advance Request Available} service invocation did not move forward the federate's logical time. The first time that the federate is allowed to use as a time stamp is its \textit{effective time}, and is equal to the current logical time plus the lookahead (Figure 4.11).
Time-Regulating Federates

Time Regulating Federates publish time stamped (TSO) data, with messages delivered in order of time-stamps.

As soon as a federate becomes time-regulating, it is subject to the federation's requirements for time stamps. The federation already promises not to allow any messages to be delivered to any federate before the value of its GALT. To ensure proper behavior after a new federate becomes time-regulating, the RTI service *Enable Time Regulation* has an implied time advance request. The logical time of the federate making the request is increased from its current value (possibly zero) to a new logical time that would prevent it from being able to use a time stamp for any message that was less than or equal to the minimum GALT of all previously existing federates. The new logical time will be communicated to the federate when the time advance is granted through the federate's *Time Advance Grant* callback routine, and will be equal to the requested time, or the current GALT minimum of other existing federates, whichever is greater. (Note that the lookahead for the federate is not used in this calculation.)

4.7 Time Advancement and Message Reception

As we mentioned earlier, a joined federate is permitted to advance its logical time during a federation execution only by requesting a time advance from the RTI by invoking one of the following services: TAR, TARA, NMR, NMRA, and FQR (see section 4.5). However, the federate's logical time shall not actually be advanced until the RTI responds with a *Time Advance Grant* service invocation at that federate. The interval between these service invocations is defined to be the Time Advancing state.

The rules governing when messages may be received depend on whether the federate is
time-constrained, and also on whether the message to be received is TSO or RO. Federates that are time-constrained may only receive TSO messages while in the Time Advancing state. RO messages may be received by a time-constrained federate only in the Time Advancing state unless the federate has previously made the Enable Asynchronous Delivery service request.

Federates that are not time-constrained may receive RO messages while in any state. Messages sent as TSO messages will be received as RO messages.

The rules governing when messages may be sent are much more complex and are provided in detail in the IEEE 1516 Federate Interface Specification (see Suggested Readings 1.). In general, the rules state that if a federate has requested a time advance to time $t_R$, it may not send a message with a time stamp less than $t_R +$ lookahead (or, less than or equal to $t_R$, if there is zero lookahead).

The Time Advance Grant service is used to grant an advance regardless of which form of request was made to advance logical time. This service shall take a logical time as an argument, and this shall be the joined federate’s new logical time (LT). The guarantee that the RTI makes about message delivery relative to the provided logical time does depend on the type of request to advance time. In some cases (when NMR, NMRA or FQR are used to request a time advancement), the RTI may advance a joined federate to a logical time LT < $t_R$ (where $t_R$ is the logical time that the federate requested). In other cases (when TAR or TARA are used to request a time advancement), the RTI shall only advance a joined federate to the logical time that was requested.

The RTI shall grant an advance to logical time LT only when it can guarantee that all TSO messages with time stamps less than LT (in some cases, less than or equal to LT) have been delivered to the joined federate. Note that in some cases, providing this guarantee shall require the RTI to wait for a significant period of wall-clock time to elapse before it can grant a time advancement to a time-constrained joined federate.

When a time-regulated federate requests to advance logical time, this acts as its guarantee not to send any TSO messages with time stamps less than time + lookahead.
Requesting Time Advancement

- Continuous Simulations
  
  To request an advance in time, use the RTI service: 
  
  *Request Time Advance.*
  
  RTI will notify when it’s ok to advance time by calling: 
  
  *Time Advance Grant.*

- Discrete event simulations
  
  The RTI service: *Next Event Request (t1)*, requests time advancement to time of next event, or to t1, whichever occurs first.

  RTI will notify when to advance time by calling: *Time Advance Grant*, and will specify the amount of the granted time advance.

---

**Figure 4.12 Requesting Time Advancement**

A federate implementing a discrete-event simulation does not have a known, or predictable, time step. Instead of making an explicit time advance request, a discrete-event simulation may call the RTI service: *Next Message Request*. This requests that the federate's logical time be increased to the time of the next message, or to the time specified as an argument if no messages will be received before this time. Like the *Time Advance Request*, this call guarantees that no future messages will be sent with time stamps before the specified time plus the lookahead.
If a federate both receives TSO data (in order of the time stamp) and sends TSO data, then that federate must declare itself to be both time regulating and time constrained (Figure 4.13). This federate is then both constrained to not advance time beyond the GALT, and regulating. As a regulating federate, it must not attempt to send data before its lookahead period (Figure 4.14).

When does a federate receive its messages? The answer depends on several factors. If the joined federate is not time-constrained, it may receive message while in any state (although, of course, only RO messages may be received). If the joined federate is time-constrained, it shall normally receive messages only in the time-advancing state.

However, joined federates may enable asynchronous message delivery (via the Enable Asynchronous Delivery service), which permits them to receive RO messages (but not TSO messages) when not in the time-advancing state.
Time Regulating and Time Constrained

- If a Federate sends and receives TSO data, in TSO order, it must be both time regulating and time constrained.
  - **Time constrained**: RTI prevents this federate from advancing time until it has received all messages that may be sent by other federates up to the requested time.
  - **Time regulating**: RTI prohibits other federates from advancing time until this federate has sent all the data that it is going to send before the requested time.

Figure 4.14 Time Regulating and Time Constrained

In general, is a joined federate is eligible to receive RO messages, it may receive all RO messages that it has not yet received. Which TSO messages will be received when a joined federate is eligible to receive TSO messages depends on which TSO messages have been sent that will be received as TSO messages, what time stamps the messages have, and what form of time advancement was requested. Since messages are not always eligible for delivery, the RTI shall internally queue pending messages for each joined federate. When the messages are finally delivered, they shall be removed from the queue.
When Are Messages Received?

Messages are only received when in a time-advancing state.

- A Federate is put into a time-advancing state by:
  - Time Advance Request OR
  - Next Event Request

- To enable receipt of RO messages at other times:
  - Enable Asynchronous Delivery (Prevents excessive delay for urgent events)

Figure 4.15 When Are Messages Received?

4.8 The HelloWorld Federate

The HelloWorld federate makes the Enable Time Constrained and the Enable Time Regulation RTI service calls, and as a result is both a time-constrained and a time-regulated federate. Time constraint is necessary for the federate to receive time-stamped messages in order of increasing time, and time regulation is necessary to send time-stamped messages to other federates.

The Enable Asynchronous Delivery RTI service is called to allow the delivery of RO events while in the simulation processing part of the program as well as when the federate is in the time-advancing state. These messages will only be received when the federate relinquishes control to the RTI by calling the function tick. Note that tick is a part of the a specific implementation of the RTI, not a part of the standard.

The lookahead value for this federate is one, although the timestep is ten! This results in an offset of one time unit in the values of time used at each time step between two federates running in the federation.
Assignment

Use the HLA Course Lab Notes to help you to create a new, passive federate, the UNFederate, that collects and displays data from any running HelloWorld federates.

Suggested Readings


2. Read Chapters 3, 5, 7, and 8 of the High Level Architecture Run-Time Infrastructure Programmer’s Guide. Also look up the interface specifications for the interfaces used in this lesson in the Appendices to this Guide. Compare the information to that obtained in [1] above. The Guide gives the actual C++ prototypes for the RTI services, for a particular implementation of the RTI. Details of operation are provided here that are not possible to specify in the implementation-independent descriptions of the services provided by the standard.