CSCI 8220 Parallel & Distributed Simulation

PDES: Distributed Virtual Environments

Introduction

High Level Architecture

Outline

- High Level Architecture (HLA): Background
- Rules
- Interface Specification
  - » Overview
  - » Class Based Subscription
  - » Attribute updates

HLA: Motivation

Department of Defense plagued by "stovepipe simulations": individual simulations designed and tailored for a specific application

- Not easily adapted for other uses, resulting in limited software reuse, much duplication of effort
- Cannot easily exploit capabilities developed in other DoD modeling and simulation programs

Goal of the High Level Architecture: define a common simulation infrastructure to support interoperability and reuse of defense simulations

- Analytic simulations (e.g., war games)
- Training (platform-level, command-level)
- Test and Evaluation

Distributed Simulation in the DoD

- SIMNET (SIMulator NETworking) (1983-89)
  - » DARPA and U.S. Army project
  - » networked interactive combat simulators
  - » tens to a few hundreds of simulators
- DIS (Distributed Interactive Simulation) (1990-96)
  - » rapid expansion based on SIMNET success
  - » tens of thousands of simulated entities
  - » IEEE standard
- Aggregate Level Simulation Protocol (ALSP) (late 1980's and 1990's)
  - » application of the networked simulations concept to war gaming models

HLA Development Process

- 10/93-1/95:three architecture proposals developed in industry
- 3/95: DMSO forms the Architecture Management Group (AMG)
- 3/95-8/96: development of baseline architecture
  - » AMG forms technical working groups (IFSpec, time management, data distribution management)
  - » Run-Time Infrastructure (RTI) prototypes
  - » prototype federations: platform level training, command level training, engineering test and evaluation, analytic analysis
- 8/96-9/96: adoption of the baseline architecture
  - » approval by AMG, Executive Council for Modeling and Simulation (EXCIMS), U.S. Under Secretary of Defense (Acquisition and Technology)
  - » 10 September, 1996: Baseline HLA approved as the standard technical architecture for all U.S. DoD simulations
- 9/96-present: continued development and standardization
  - » Varying levels of adoption
  - » Commercialization of RTI software
  - » Standardization (IEEE 1516)

High Level Architecture (HLA)

Background:

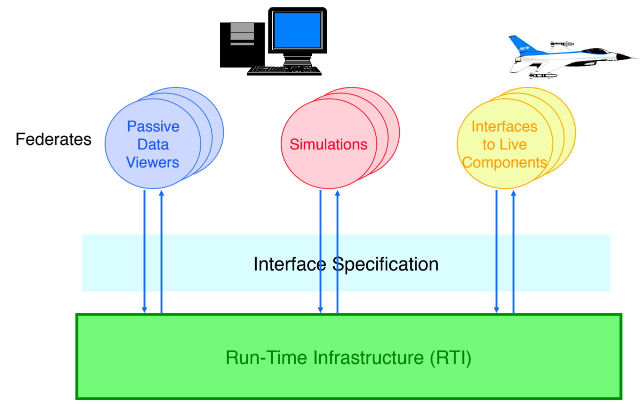
- Based on a composable "system of systems" approach
  - » no single simulation can satisfy all user needs
  - » support interoperability and reuse among DoD simulations
- Federations of simulations (federates)
  - » pure software simulations
  - » human-in-the-loop simulations (virtual simulators)
  - » live components (e.g., instrumented weapon systems)

High Level Architecture (HLA)

The HLA consists of

- **Rules** that simulations (federates) must follow to achieve proper interaction during a federation execution
- **Object Model Template (OMT)** defines the format for specifying the set of common objects used by a federation (federation object model), their attributes, and relationships among them
- **Interface Specification (IFSpec)** provides interface to the Run-Time Infrastructure (RTI), that ties together federates during model execution

An HLA Federation



Federation Rules

1. Federations shall have an HLA Federation Object Model (FOM), documented in accordance with the HLA Object Model Template (OMT).
2. In a federation, all simulation-associated object instance representation shall be in the federates, not in the runtime infrastructure (RTI).
3. During a federation execution, all exchange of FOM data among joined federates shall occur via the RTI.
4. During a federation execution, joined federates shall interact with the RTI in accordance with the HLA interface specification.
5. During a federation execution, an instance attribute shall be owned by at most one federate at any given time.

Federate Rules (cont)

6. Federates shall have an HLA Simulation Object Model (SOM), documented in accordance with the HLA Object Model Template (OMT).
7. Federates shall be able to update and/or reflect any instance attributes and send and/or receive interactions, as specified in their SOM.
8. Federates shall be able to transfer and/or accept ownership of instance attributes dynamically during a federation execution, as specified in their SOMs.
9. Federates shall be able to vary the conditions (e.g., thresholds) under which they provide updates of instance attributes, as specified in their SOM.
10. Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation.

Interface Specification

Category	Functionality
Federation Management	Create and delete federation executions join and resign federation executions control checkpoint, pause, resume, restart
Declaration Management	Establish intent to publish and subscribe to object attributes and interactions
Object Management	Create and delete object instances Control attribute and interaction publication Create and delete object reflections
Ownership Management	Transfer ownership of object attributes
Time Management	Coordinate the advance of logical time and its relationship to real time
Data Distribution Management	Supports efficient routing of data

Message Passing Alternatives

- Traditional message passing mechanisms: Sender explicitly identifies receivers
  - Destination process, port, etc.
  - Poorly suited for federated simulations
- Broadcast
  - Receiver discards messages not relevant to it
  - Used in SIMNET, DIS (initially)
  - Doesn't scale well to large federations
- Publication / Subscription mechanisms
  - Analogous to newsgroups
  - Producer of information has a means of describing data it is producing
  - Receiver has a means of describing the data it is interested in receiving
  - Used in High Level Architecture (HLA)

A Typical Federation Execution

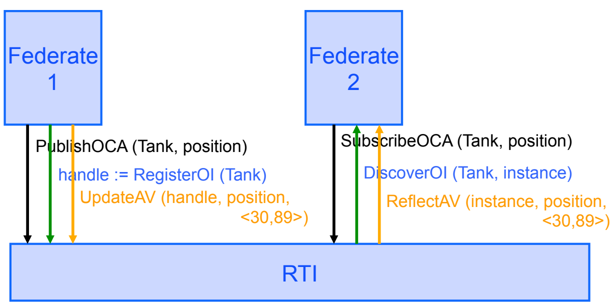
1. Initialize federation
   » Create Federation Execution (Federation Mgt)
   » Join Federation Execution (Federation Mgt)
2. Declare objects of common interest among federates
   » Publish Object Class Attributes (Declaration Mgt)
   » Subscribe Object Class Attributes (Declaration Mgt)
3. Exchange information
   » Update/Reflect Attribute Values (Object Mgt)
   » Send/Receive Interaction (Object Mgt)
   » Time Advance Request, Time Advance Grant (Time Mgt)
   » Request Attribute Ownership Assumption (Ownership Mgt)
   » Send Interaction with Regions (Data Distribution Mgt)
4. Terminate execution
   » Resign Federation Execution (Federation Mgt)
   » Destroy Federation Execution (Federation Mgt)

Class-Based Data Distribution

- Federation Object Model (FOM) defines type of information transmitted among federates
  » Object classes (e.g., tank)
  » Attributes (e.g., position, orientation of turret)
- A few key primitives (Federate/RTI interface)
  » Publish Object Class Attributes: Called by a federate to declare the object classes and attributes it is able to update
  » Subscribe Object Class Attributes: Declare the object classes and attributes that the federate is interested in receiving
  » Register Object Instance: Notify RTI an *instance* of an object has been created within the federate
  » Discover Object Instance*: Notify federate an instance of an object of a subscribed class has been registered
  » Update Attribute Values: notify RTI one or more attributes of an object has been modified
  » Reflect Attribute Values*: notify federate attributes to which it has subscribed have been modified

* Denotes callback from RTI to federate

Example



OCA = Object Class Attributes
OI = Object Instance
AV = Attribute Values

Summary

- The High Level Architecture is an example of an approach for realizing distributed simulations
- HLA Rules define general principles that pervade the entire architecture
- HLA Interface Specification defines a set of run-time services to support distributed simulations
- Data distribution is based on a publication / subscription mechanism

PDES: Distributed Virtual Environments
Time Management **in the High Level Architecture**

Outline

- Overview of time management services
- Time constrained and time regulating federates
- Related object management services
- Time Advance Request (TAR)
- Next Event Request (NER)
- Lookahead

HLA Message Order Services

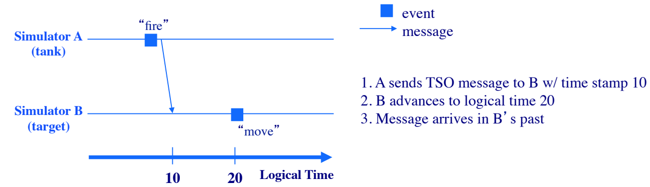
- **Receive order (*RO*):** *Messages passed to federate in an arbitrary order (unordered)*
- ***Time stamp order (TSO):*** Sender assigns a time stamp to message; successive messages passed to each federate have non-decreasing time stamps

Property	*RO*	*TSO*
Latency	*low*	*higher*
reproduce before and after relationships?	*no*	*yes*
all federates see same ordering of events?	*no*	*yes*
execution repeatable?	*no*	*yes*
typical applications	*training, T&E*	*analysis*

- *receive order minimizes latency, does not prevent temporal anomalies*
- *TSO prevent temporal anomalies, but has somewhat higher latency*

Time Synchronized Delivery

Consider interconnecting two sequential, discrete event simulators

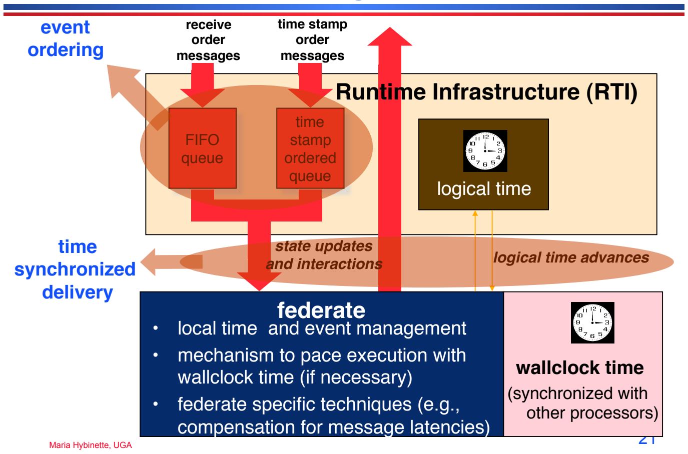


In the HLA, logical time is synonymous with simulation time

Logical time advances by each simulator must be properly managed to ensure no simulator receives a message in its past.

HLA Time Management (TM) services define a protocol for federates to advance their logical time; RTI will ensure TSO messages are not delivered in a federate's past

HLA Time Management Services



Time Regulating & Time Constrained Federates

Federates must declare their intent to utilize time management services by setting their *time regulating* and/or *time constrained* flags

- **Time regulating federates: can send TSO messages**
  - **Can prevent other federates from advancing their logical time**
  - **Enable Time Regulation … Time Regulation Enabled †**
  - **Disable Time Regulation**
- **Time constrained federates: can receive TSO messages**
  - **Time advances are constrained by other federates**
  - **Enable Time Constrained … Time Constrained Enabled †**
  - **Disable Time Constrained**
- **Each federate in a federation execution can be**
  - **Time regulating only (e.g., message source)**
  - **Time constrained only (e.g., Stealth)**
  - **Both time constrained and regulating (common case for analytic simulations)**
  - **Neither time constrained nor regulating (e.g., DIS-style training simulations)**

† indicates callback to federate

Related Object Management Services

Sending and Receiving Messages

- **Update Attribute Values … Reflect Attribute Values †**
- **Send Interaction … Receive Interaction †**

Message Order (*Receive Order or Time Stamp Order*)

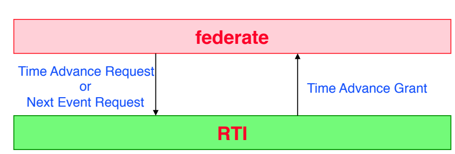
- ***Preferred Order Type:* default order type specified in "fed file" for each attribute and interaction**
- ***Sent Message Order Type:***
  - **TSO if preferred order type is TSO and the federate is time regulating and a time stamp was used in the *Update Attribute Values* or *Send Interaction* call**
  - **RO otherwise**
- **Received Message Order Type**
  - **TSO if sent message order type is TSO and receiver is time constrained**
  - **RO otherwise**

† indicates callback to federate

HLA Time Management (TM) Services

HLA TM services define a protocol for federates to advance logical time; logical time only advances when that federate explicitly requests an advance

- **Time Advance Request: *time stepped federates***
- **Next Event Request: *event stepped federates***
- **Time Advance Grant: RTI invokes to acknowledge logical time advances**

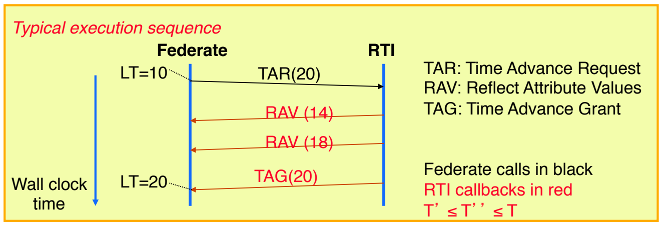


If the logical time of a federate is T, the RTI guarantees no more TSO messages will be passed to the federate with time stamp < T

Federates responsible for pacing logical time advances with wallclock time in real-time executions

Time Advance Request (TAR)

- Typically used by time stepped federates
- Federate invokes Time Advance Request (T) to request its logical time (LT) be advanced to T
- RTI delivers all TSO messages with time stamp ≤ T
- RTI advances federate's time to T, invokes Time Advance Grant (T) when it can guarantee all TSO messages with time stamp ≤ T have been delivered
- Grant time always matches the requested time



Code Example: Time Stepped Federate

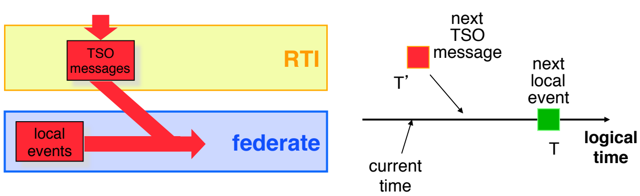
sequential simulator	*federated simulator*

T = current simulation time
While (simulation not complete)
  update local simulation state
  T = T + ΔT;
End-While
	While (simulation not complete)
	  update local simulation state
	  UpdateAttributeValues (…)
	  PendingTAR = TRUE;
	  TimeAdvanceRequest(T+ ΔT)
	  while (PendingTAR) Tick*(…);
	  T = T + ΔT;
	End-While
	
	/* the following federate-defined procedures are called by the RTI */
	Procedure ReflectAttributeValues (…)
	  update local state
	
	Procedure TimeAdvanceGrant (…)
	  PendingTAR = False;

* Tick is only used in single threaded RTI implementations

Next Event Request (NER)

- Typically used by event stepped federates
- Goal: process all events (local and incoming TSO messages) in time stamp order



Federate: next local event has time stamp T

- If no TSO messages w/ time stamp < T, advance to T, process local event
- If there is a TSO message w/ time stamp T' ≤ T, advance to T' and process TSO message

Next Event Request (NER)

Federate invokes Next Event Request (T) to request its logical time be advanced to time stamp of next TSO message, or T, which ever is smaller

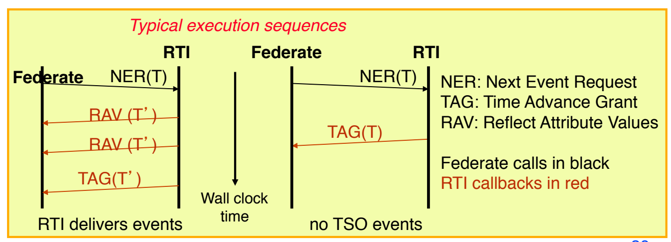
If next TSO message has time stamp T' ≤ T

RTI delivers next TSO message, and all others with time stamp T'

RTI issues Time Advance Grant (T')

Else

RTI advances federate's time to T, invokes Time Advance Grant (T)

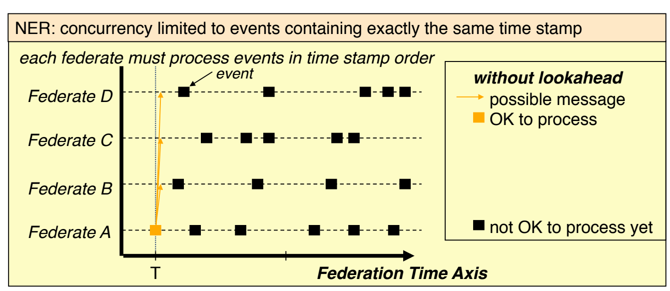


Code Example: Event Stepped Federate

sequential simulator	*federated simulator*

T = current simulation time
PES = pending event set
 	
While (simulation not complete)
  T = time of next event in PES
  process next event in PES
End-While
	While (simulation not complete)
	  T = time of next event in PES
	  PendingNER = TRUE;
	  NextEventRequest(T)
	  while (PendingNER) Tick(…);
	  process next event in PES
	End-While
	
	/* the following federate-defined procedures are called by the RTI */
	Procedure ReflectAttributeValues (…)
	  place event in PES
	
	Procedure TimeAdvanceGrant (…)
	  PendingNER = False;

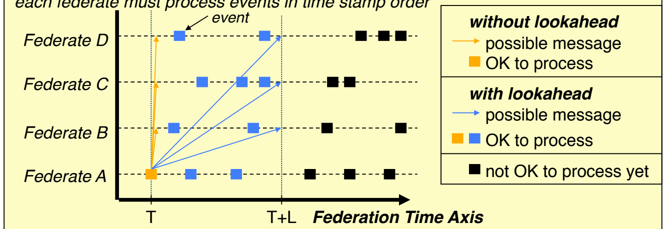
Lookahead



Lookahead

NER: concurrency limited to events containing exactly the same time stamp

each federate must process events in time stamp order

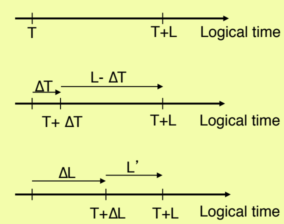


Each federate using logical time declares a lookahead value L; any TSO message sent by the federate must have a time stamp ≥ the federate's current time + L

Lookahead is necessary to allow concurrent processing of events with different time stamps (unless optimistic event processing is used)

Lookahead in the HLA

- Each federate must declare a non-negative lookahead value
- Any TSO sent by a federate must have time stamp at least the federate's current time plus its lookahead
- Lookahead can change during the execution (*Modify Lookahead*)
  - » increases take effect immediately
  - » decreased do not take effect until the federate advances its logical time



1. Current time is T, lookahead L
2. Request lookahead decrease by ΔL to L'
3. Advance ΔT, lookahead, decreases ΔT
4. After advancing ΔL, lookahead is L'

Federate/RTI Guarantees

Federate at logical time T (with lookahead L)

- **All outgoing TSO messages must have time stamp ≥ T+L (L>0)**

Time Advance Request (T)

- **Once invoked, federate cannot send messages with time stamp less than T plus lookahead**

Next Event Request (T)

- **Once invoked, federate cannot send messages with time stamp less than T plus the federate's lookahead unless a grant is issued to a time less than T**

Time Advance Grant (T) (after TAR or NER service)

- **All TSO messages with time stamp less than or equal to T have been delivered**

Summary

- **HLA time management designed to support interoperability of simulations with different time advance mechanisms**
  - » **Time stepped federates**
  - » **Event-driven federates**
- **Time management services include services to order messages (time stamp ordered delivery) and mechanisms to advance simulation time**
- **Time regulating/constrained used to "turn on" time management**
- **Per federate lookahead supported**