Outline

• Introduction
• Distributed Interactive Simulation
• Dead Reckoning
• High Level Architecture
Geographically Distributed Users/Resources

Geographically distributed users and/or resources are sometime needed
- Interactive games over the Internet
- Specialized hardware or databases

Server architecture

Cluster of workstations on LAN

Distributed architecture

Distributed computers

WAN interconnect

LAN interconnect
Distributed Interactive Simulation (DIS)

“The primary mission of DIS is to define an infrastructure for linking simulations of various types at multiple locations to create realistic, complex, virtual ‘worlds’ for the simulation of highly interactive activities” [DIS Vision, 1994].

- developed in U.S. Department of Defense, initially for training
- distributed virtual environments widely used in DoD; growing use in other areas (entertainment, emergency planning, air traffic control)
**DIS Design Principles**

- **Autonomy of simulation nodes**
  - simulations broadcast events of interest (messages are called Protocol Data Units PDU) to other simulations; need not determine which others need information
  - receivers determine if information is relevant to it, and model local effects of new information
  - simulations may join or leave exercises in progress

- **Transmission of “ground truth” information**
  - each simulation transmits absolute truth about state of its objects
  - receiver is responsible for appropriately “degrading” information (e.g., due to environment, sensor characteristics)

- **Transmission of state change information only**
  - if behavior “stays the same” (e.g., straight and level flight), state updates drop to a predetermined rate (e.g., every five seconds)

- **“Dead Reckoning” algorithms**
  - extrapolate current position of moving objects based on last reported position

- **Simulation time constraints**
  - many simulations are human-in-the-loop
  - humans cannot distinguish temporal difference < 100 milliseconds
  - places constraints on communication latency of simulation platform
A Typical DVE Node Simulator

Execute every 1/30th of a second:
• receive incoming messages & user inputs, update state of remote vehicles
• update local display
• for each local vehicle
  • compute (integrate) new state over current time period
  • send messages (e.g., broadcast) indicating new state

Typical Sequence

1. Detect trigger press
2. Audio “fire” sound
3. Display muzzle flash
4. Send fire PDU
5. Display muzzle flash
6. Compute trajectory, display tracer
7. Display shell impact
8. Send detonation PDU
9. Display shell impact
10. Compute damage
11. Send Entity state PDU indicating damage
Distributed Simulation Example

- Virtual environment simulation containing two moving vehicles
- One vehicle per federate (simulator)
- Each vehicle simulator must track location of other vehicle and produce local display (as seen from the local vehicle)
- Approach 1: Every 1/60th of a second:
  - Each vehicle sends a message to other vehicle indicating its current position
  - Each vehicle receives message from other vehicle, updates its local display
Limitations

- Position information corresponds to location when the message was sent; doesn’t take into account delays in sending message over the network
- Requires generating many messages if there are many vehicles
Dead Reckoning

- Send position update messages less frequently
- Local dead reckoning model predicts the position of remote entities between updates

When are updates sent?
- How does the DRM predict vehicle position?
When are position update messages generated?
- Compare DRM position with exact position, and generate an update message if error is too large
- Generate updates at some minimum rate, e.g., 5 seconds (heart beats)
Dead Reckoning Models

• $P(t) = \text{precise position of entity at time } t$
• Position update messages: $P(t_1), P(t_2), P(t_3) \ldots$
• $v(t_i), a(t_i) = \text{ith velocity, acceleration update}$
• DRM: estimate $D(t)$, position at time $t$
  
  $t_i = \text{time of last update preceding } t$
  
  $\Delta t = t_i - t$

• Zeroth order DRM:
  
  $D(t) = P(t_i)$

• First order DRM:
  
  $D(t) = P(t_i) + v(t_i)\Delta t$

• Second order DRM:
  
  $D(t) = P(t_i) + v(t_i)\Delta t + 0.5a(t_i)(\Delta t)^2$
DRM Example

Potential problems:
• Discontinuity may occur when position update arrives; may produce “jumps” in display
• Does not take into account message latency
Time Compensation

Taking into account message latency

- Add time stamp to message when update is generated (sender time stamp)
- Dead reckon based on message time stamp

![Diagram](image)

- true position
- state update
- message
- DRM estimate of true position
- display update

(update with time compensation)
Smoothing

Reduce discontinuities after updates occur
• “phase in” position updates
• After update arrives
  – Use DRM to project next k positions
  – Interpolate position of next update

Accuracy is reduced to create a more natural display
High Level Architecture (HLA)

- 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)
- mandated as the standard reference architecture for all M&S in the U.S. Department of Defense (September 1996)

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
HLA Speak

Simulation model/package = Federate

Combination of simulation models = Federation
HLA

• High Level Architecture (HLA) provides a common structure for modelling and simulation within DoD to facilitate *interoperability* among simulations and to promote *reuse* of simulations and their components.

• HLA is now an IEEE open standard (1516) (Oct 2000)

• HLA rules
• HLA interface specification
• HLA object model template
HLA

- Federation Rules
  - Ensure proper interaction of simulations in federation
  - Describe the simulation and federate responsibilities
- Interface Specification
  - Defines Run time infrastructure (RTI) services
  - Identifies “callback” functions each federate must provide
- Object Model Template
  - Provides a common method for recording information
  - Establishes the format of key models
    - Federation Object Model (FOM)
    - Simulation Object Model (SOM)
Interconnecting autonomous simulators

Runtime Infrastructure (RTI)
Services to create and manage the execution of the federation
- Federation setup / tear down
- Transmitting data among federates
- Synchronization (time management)
Functional View of HLA

- Support Utilities
- Simulations
- Live Player Interfaces

Runtime Infrastructure
- Federation Management
- Declaration Management
- Object Management
- Ownership Management
- Time Management
- Data Distribution Management

Live Participants
HLA Runtime Infrastructure

• Runtime Infrastructure (RTI) is a collection of software that provides common services required by multiple simulation systems.

• Six service groups:
  – Federation Management
  – Declaration Management
  – Object Management
  – Ownership Management
  – Time Management
  – Data Distribution Management
## Interface Specification

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

**initialize federation**
- Create Federation Execution (Federation Mgt)
- Join Federation Execution (Federation Mgt)

**declare objects of common interest among federates**
- Publish Object Class (Declaration Mgt)
- Subscribe Object Class Attribute (Declaration Mgt)

**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)
- Modify Region (Data Distribution Mgt)

**terminate execution**
- Resign Federation Execution (Federation Mgt)
- Destroy Federation Execution (Federation Mgt)
Federated vs. RTI Initiated Services

Some services are initiated by the federate, others by the RTI

- **Federate invoked services**
  - Publish, subscribe, register, update
  - Not unlike calls to a library
  - Procedures defined in the RTI ambassador

- **RTI invoked services**
  - Discover, reflect
  - Federate defined procedures, in Federate Ambassador