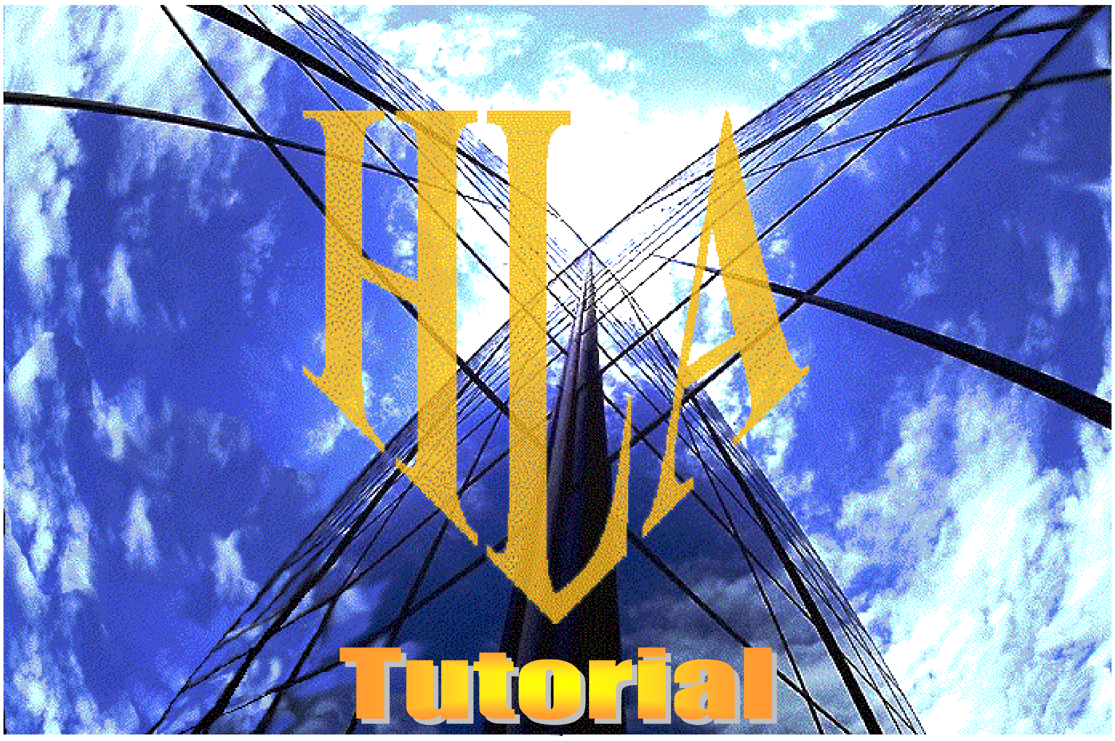
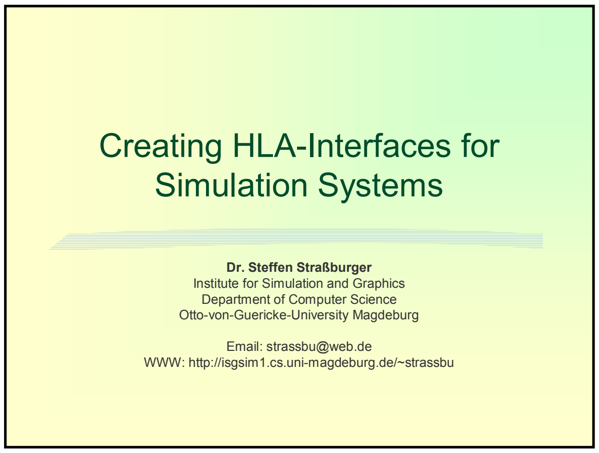
2001 European Simulation Interoperability Workshop
University of Westminster, Harrow Campus
June 25-27, 2001



Creating HLA Interfaces for Simulation Systems

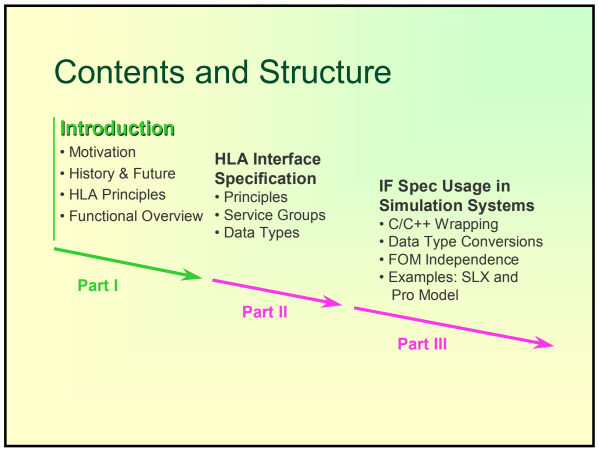
Steffen Strassburger
Institute for Simulation and Graphics
Department of Computer Science
Otto-von-Guericke-University Magdeburg, Germany
strassbu@web.de



Creating HLA-Interfaces for Simulation Systems

Dr. Steffen Straßburger
Institute for Simulation and Graphics
Department of Computer Science
Otto-von-Guericke-University Magdeburg

Email: strassbu@web.de
WWW: http://isgsim1.cs.uni-magdeburg.de/~strassbu



Contents and Structure

Introduction
- Motivation
- History & Future
- HLA Principles
- Functional Overview

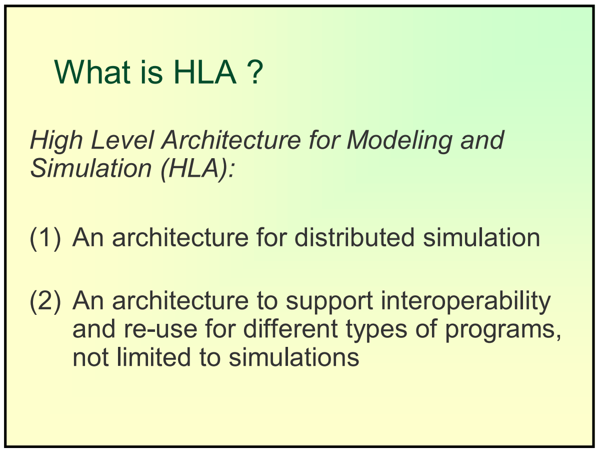
Part I

HLA Interface Specification
- Principles
- Service Groups
- Data Types

Part II

IF Spec Usage in Simulation Systems
- C/C++ Wrapping
- Data Type Conversions
- FOM Independence
- Examples: SLX and Pro Model

Part III



What is HLA ?

High Level Architecture for Modeling and Simulation (HLA):

(1) An architecture for distributed simulation

(2) An architecture to support interoperability and re-use for different types of programs, not limited to simulations

Motivation: Why HLA ?

- No single monolithic simulation can satisfy the needs of all users.
- All uses of simulations and useful ways of combining them cannot be anticipated in advance.
- Consequence:
  - necessity of a modular / composable approach to constructing simulation federations
- US DoD approach: The High Level Architecture
  - Simulation functionality separated from general purpose interoperability support infrastructure

HLA: History and Future

- DoD Modeling & Simulation Master Plan 1995:

  „Establish a common high-level simulation architecture to facilitate the interoperability of all types of models and simulations ....,as well as to facilitate the reuse of M&S components"
- Combines predecessor technologies (DIS, ALSP)
- HLA has passed IEEE standardization and is the future standard architecture for all DoD simulations

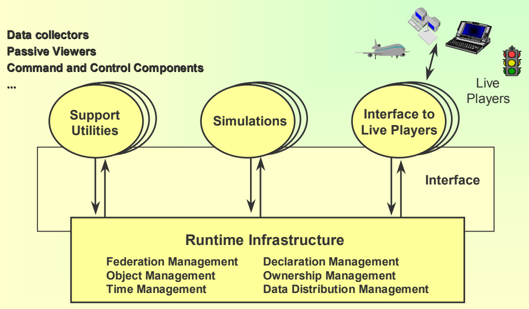
Principles of the HLA approach (1)

- Interoperability
  - Simulations must be able to exchange data and meaningful interpret them
- Reusability
  - Simulations must have well-defined and well-documented interfaces and objects
- Federations of Simulations
  - Different simulations (federates) together form a federation (federation execution)

Principles of the HLA approach (2)

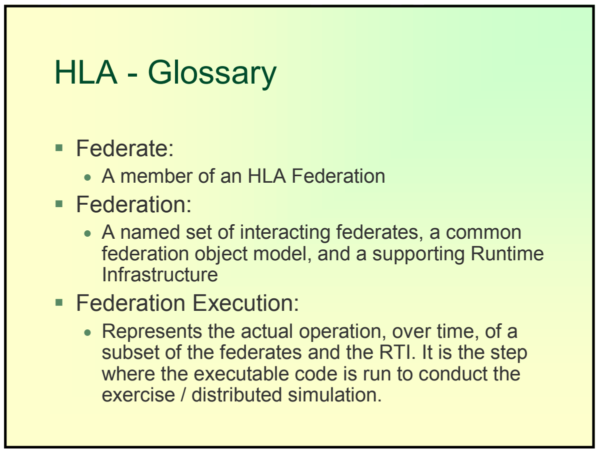
- Differentiation between simulation functionality and basic services
  - All basic services (data exchange, communication) are to be provided by a Runtime Infrastructure (RTI) which interfaces with the simulations in a well-defined way
- Object view on simulations
  - All modeled entities are considered “Objects”
    - Attributes for modeling their characteristic and condition
    - Interactions for modeling communication between objects
    - No restriction on implementation *inside* simulations

Functional Overview of HLA



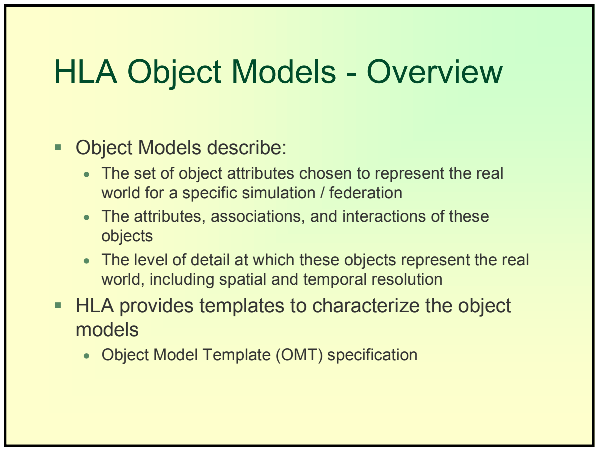
HLA - the key defining elements

- HLA Rules
  - define the cooperation of simulations
- HLA Object Model Template
  - defines an object view on the simulations
- HLA Interface Specification
  - Application Programming Interface that all simulations have to comply with
  - All communication between simulations is only allowed via this interface



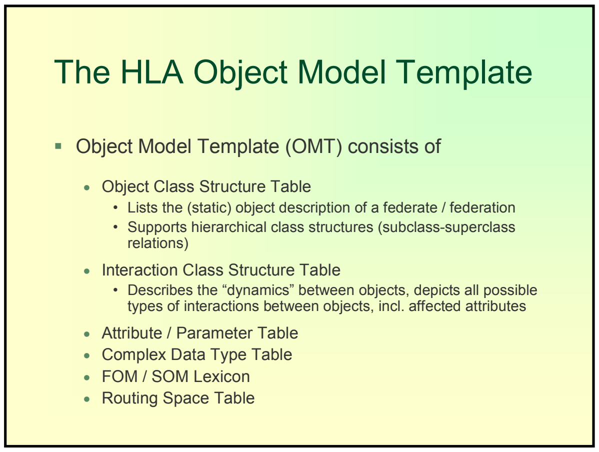
HLA - Glossary

- Federate:
  - A member of an HLA Federation
- Federation:
  - A named set of interacting federates, a common federation object model, and a supporting Runtime Infrastructure
- Federation Execution:
  - Represents the actual operation, over time, of a subset of the federates and the RTI. It is the step where the executable code is run to conduct the exercise / distributed simulation.



HLA Object Models - Overview

- Object Models describe:
  - The set of object attributes chosen to represent the real world for a specific simulation / federation
  - The attributes, associations, and interactions of these objects
  - The level of detail at which these objects represent the real world, including spatial and temporal resolution
- HLA provides templates to characterize the object models
  - Object Model Template (OMT) specification



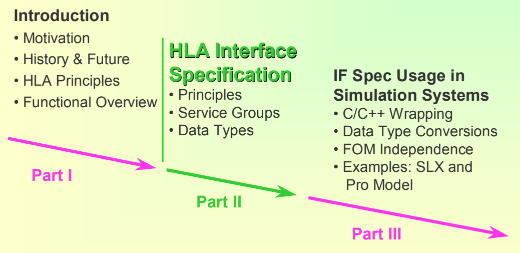
The HLA Object Model Template

- Object Model Template (OMT) consists of
  - Object Class Structure Table
    - Lists the (static) object description of a federate / federation
    - Supports hierarchical class structures (subclass-superclass relations)
  - Interaction Class Structure Table
    - Describes the “dynamics” between objects, depicts all possible types of interactions between objects, incl. affected attributes
  - Attribute / Parameter Table
  - Complex Data Type Table
  - FOM / SOM Lexicon
  - Routing Space Table

HLA Definition: The FOM and SOM's

- Simulation Object Model (SOM)
  - describes a federate's modeling capabilities/characteristics in terms of object classes, interactions, attributes, parameters, ownership transfer capabilities, etc.
  - follows the guidelines established in the OMT-description
- Federation Object Model (FOM)
  - Contract among "n" simulations to satisfy the objectives of a specific federation
  - FOM content gives a description of all shared information

Contents and Structure


Introduction
- Motivation
- History & Future
- HLA Principles
- Functional Overview

HLA Interface Specification
- Principles
- Service Groups
- Data Types

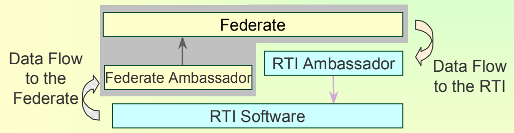
IF Spec Usage in Simulation Systems
- C/C++ Wrapping
- Data Type Conversions
- FOM Independence
- Examples: SLX and Pro Model

The HLA Interface Specification

- Definition of the interface services between the Runtime Infrastructure and the simulations
  - 6 service groups
- API's (Application Programming Interfaces) for different language bindings following the general Interface Specification
  - C++
  - Java
  - ADA95
  - CORBA IDL

The RTI-Federate Communication

- Communication through ambassadors objects



- RTI-Ambassador for calls from the federate to the RTI
  - provided in a library that has to be linked to the federate
- Federate Ambassador for RTI calls to the federate
  - To be implemented by the federate (C++: abstract object class)

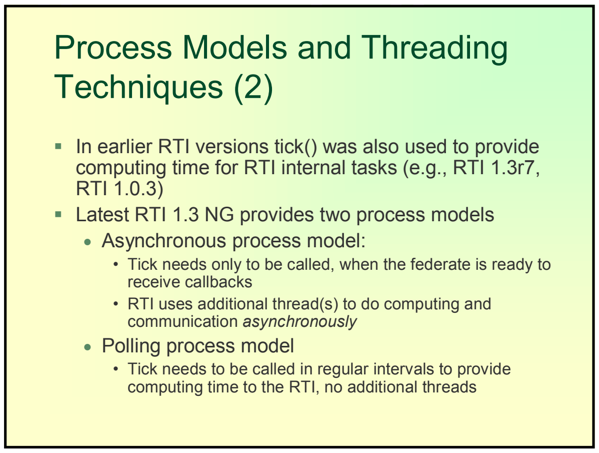
Process Models and Threading Techniques (1)

- Callback handling plays important role
  - A federate calls methods of the RTI ambassador
  - Callbacks from the RTI to the federate ambassador must be *triggered* by calling a special method called *tick()*
  - Most RTI ambassador methods are not re-entrant, i.e., you cannot call an RTI ambassador method while processing a federate ambassador callback
  - Two versions of *tick()*
    - Usage depends on the requirements of the federate
    - Return values indicates if more callbacks are pending

Two versions of *Tick()*[*]

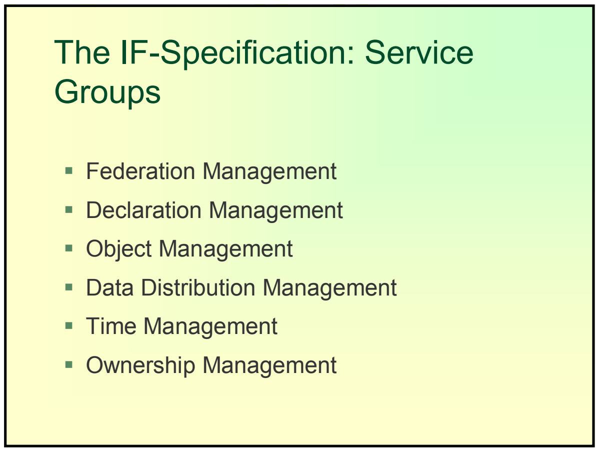
```
Boolean tick ()
throw (
  SpecifiedSaveLabelDoesNotExist,
  ConcurrentAccessAttempted,
  RTIinternalError);

Boolean tick (
       TickTime minimum,
       TickTime maximum)
throw (
  SpecifiedSaveLabelDoesNotExist,
  ConcurrentAccessAttempted,
  RTIinternalError);
```

* from C++
IF Specification



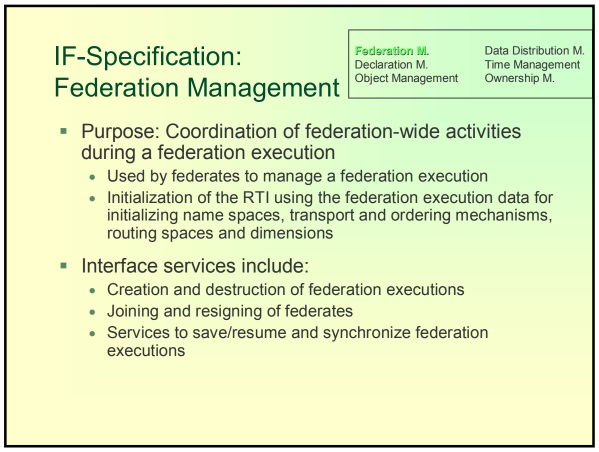
Process Models and Threading Techniques (2)

- In earlier RTI versions tick() was also used to provide computing time for RTI internal tasks (e.g., RTI 1.3r7, RTI 1.0.3)
- Latest RTI 1.3 NG provides two process models
  - Asynchronous process model:
    - Tick needs only to be called, when the federate is ready to receive callbacks
    - RTI uses additional thread(s) to do computing and communication *asynchronously*
  - Polling process model
    - Tick needs to be called in regular intervals to provide computing time to the RTI, no additional threads



The IF-Specification: Service Groups

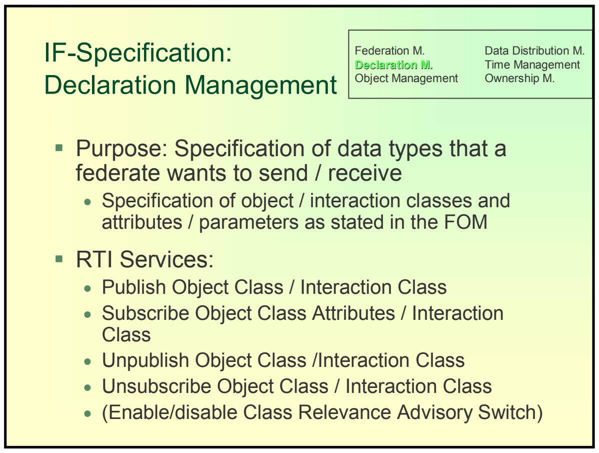
- Federation Management
- Declaration Management
- Object Management
- Data Distribution Management
- Time Management
- Ownership Management



IF-Specification: Federation Management

Federation M.	Data Distribution M.
Declaration M.	Time Management
Object Management	Ownership M.

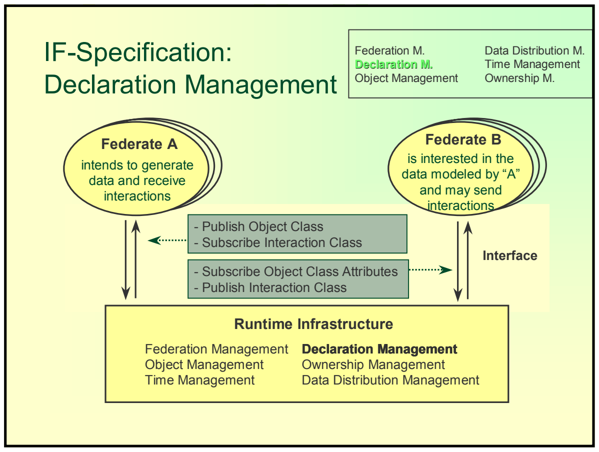
- Purpose: Coordination of federation-wide activities during a federation execution
  - Used by federates to manage a federation execution
  - Initialization of the RTI using the federation execution data for initializing name spaces, transport and ordering mechanisms, routing spaces and dimensions
- Interface services include:
  - Creation and destruction of federation executions
  - Joining and resigning of federates
  - Services to save/resume and synchronize federation executions



IF-Specification: Declaration Management

Federation M.	Data Distribution M.
Declaration M.	Time Management
Object Management	Ownership M.

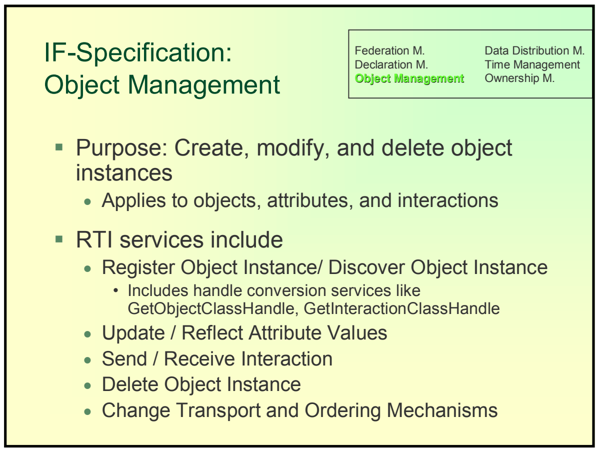
- Purpose: Specification of data types that a federate wants to send / receive
  - Specification of object / interaction classes and attributes / parameters as stated in the FOM
- RTI Services:
  - Publish Object Class / Interaction Class
  - Subscribe Object Class Attributes / Interaction Class
  - Unpublish Object Class /Interaction Class
  - Unsubscribe Object Class / Interaction Class
  - (Enable/disable Class Relevance Advisory Switch)



IF-Specification: Declaration Management

Federation M.	Data Distribution M.
Declaration M.	Time Management
Object Management	Ownership M.

Federate A
intends to generate data and receive interactions
Federate B
is interested in the data modeled by "A" and may send interactions
- Publish Object Class
- Subscribe Interaction Class
- Subscribe Object Class Attributes
- Publish Interaction Class
Interface
Runtime Infrastructure
Federation Management
Declaration Management
Object Management
Ownership Management
Time Management
Data Distribution Management



IF-Specification: Object Management

Federation M.	Data Distribution M.
Declaration M.	Time Management
Object Management	Ownership M.

- Purpose: Create, modify, and delete object instances
  - Applies to objects, attributes, and interactions
- RTI services include
  - Register Object Instance/ Discover Object Instance
    - Includes handle conversion services like GetObjectClassHandle, GetInteractionClassHandle
  - Update / Reflect Attribute Values
  - Send / Receive Interaction
  - Delete Object Instance
  - Change Transport and Ordering Mechanisms

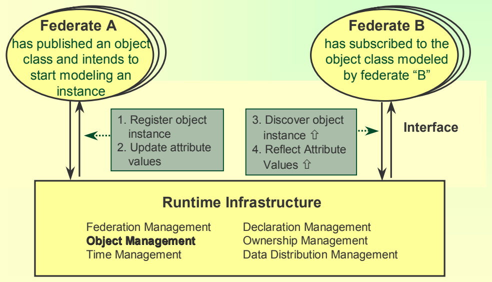
IF-Specification: Object Management

Federation M.	Data Distribution M.
Declaration M.	Time Management
Object Management	Ownership M.

- Ordering types for messages (attribute updates, interactions)
  - receive order (RO)
  - time stamp order (TSO)
- Transportation types (differ in terms of reliability)
  - reliable
  - best effort

IF-Specification: Object Management

Federation M.	Data Distribution M.
Declaration M.	Time Management
Object Management	Ownership M.


Federate A
has published an object class and intends to start modeling an instance
Federate B
has subscribed to the object class modeled by federate "B"
1. Register object instance
2. Update attribute values
3. Discover object instance ⇧
4. Reflect Attribute Values ⇧
Interface
Runtime Infrastructure
Federation Management
Object Management
Time Management
Declaration Management
Ownership Management
Data Distribution Management

IF-Specification: Data Distribution Management

Federation M.	Data Distribution M.
Declaration M.	Time Management
Object Management	Ownership M.

- Purpose: Reduce the network traffic
- Basic Concept: Routing Spaces
  - Multi-dimensional coordinate system
  - Federates can specify regions
    - into which they want to send data (Update Region)
    - from which they want to receive data (Subscription Region)
    - Data will only be transferred if Update/Subscription Regions overlap
- Federates don't have to use DDM
  - Care must be taken when mixing DDM and DM federates

IF-Specification: Data Distribution Management

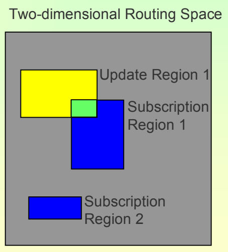
Federation M.	**Data Distribution M.**
Declaration M.	Time Management
Object Management	Ownership M.

🟨 Update Region

🟦 Subscription Region

🟩 Overlapping Region

- attributes and interactions will be sent to the subscribing federate



IF-Specification: Time Management

Federation M.	Data Distribution M.
Declaration M.	**Time Management**
Object Management	Ownership M.

- General approaches in distributed simulation:
  - conservative synchronization (with lookahead)
  - optimistic synchronization (e.g. time warp)
  - hybrid methods
  - time-stepped
  - real-time driven
  - (no coordination necessary)
- HLA claims to support all mechanisms by providing a transparent time management
  - local time management of federates is invisible to the outside

IF-Specification: Time Management

Federation M.	Data Distribution M.
Declaration M.	**Time Management**
Object Management	Ownership M.

- Federates have to *request* their local time advancement
  - Next Event Request & Time Advance Request
    - federate indicates that it will not generate new messages prior to the requested time advance (if it doesn't receive new messages)
  - RTI issues “Time Advance Grant”
    - once a time advance grant has been issued, no messages with a smaller time stamp are allowed to be sent
- RTI coordinates time advances under consideration of:
  - ordering mechanisms for attributes / interactions
  - requirements/properties of the federates

IF-Specification: Time Management

Federation M.	Data Distribution M.
Declaration M.	Time Management
Object Management	Ownership M.

- Two Switches determine the basic time management characteristics of a federate

		Time-Regulating: true	Time-Regulating: false
Time-Constrained	true	Strictly time synchronized: conservative (ALSP), aggressive (Time Warp)	Viewer / Federation Management Tool: stays synchronized, but does not generate events
Time-Constrained	false	unconstrained (DIS) cooperation with conservative federates	Externally synchronized Simulation: no RTI-based Time Management (DIS)

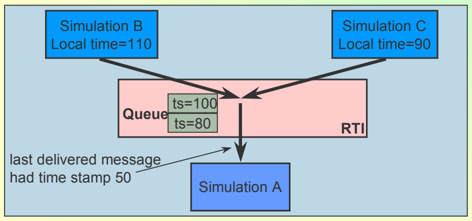
IF-Specification: Time Management Example

Federation M.	Data Distribution M.
Declaration M.	Time Management
Object Management	Ownership M.

```
while (simulation still in progress) {
    Determine timestamp of next local event, let TS_local be this timestamp
    /* next statement enables delivery of next external message */
    invoke **Next Event Request** (TS_local) service
    honor zero or more RTI requests for **Reflect Attribute Value** and **Receive
            Interaction** services
    honor RTI service request for **Time Advance Grant**
    if (no TimeStampOrdered messages received in above RTI service requests)
    {
            now = TS_local
            process the next local event identified above
            }
    else {
            now = timestamp of TimeStampOrdered message
            process message;
            }
    provide any changed information (new attribute values or interactions)
            to the RTI via the **Update Attribute** and/or **Send Interaction** services.
    }
```

IF-Specification: Time Management

Federation M.	Data Distribution M.
Declaration M.	Time Management
Object Management	Ownership M.



- Can the RTI deliver the messages in the queue to Simulation A ?
- RTI offers a mechanism to ensure the time stamp order (no events in the past (with a lower time stamp than previously delivered events) will be delivered

IF-Specification: Time Management

Federation M.	Data Distribution M.
Declaration M.	Time Management
Object Management	Ownership M.

- Dealing with optimistic federates
  - Event Retraction Handles
    - Returned by Update Attribute Values and Send Interaction services
    - Can be used to send “anti-messages” to cancel the event/message
  - Optimistic federates can exchange optimistic messages
    - Have to request “optimistic” delivery of messages to them
    - RTI service “Flush Queue Request”
- RTI prevents conservative federates from receiving optimistic messages
  - Only optimistic federates need to know how to “rollback”

IF-Specification: Time Management

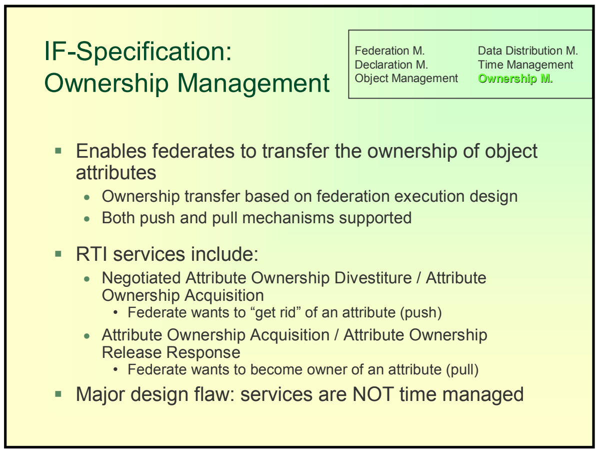
Federation M.	Data Distribution M.
Declaration M.	Time Management
Object Management	Ownership M.

- HLA Time Management does not define a generally valid federation time
  - “The federation time is X” is not an valid statement
  - “The federation time is X from the point of view of federate Y” is a valid statement
- “Lower Bound Time Stamp” of a federation:
  - Minimum time stamp such that it can be guaranteed that no federate will generate any time-stamp-ordered events with a lower time stamp

IF-Specification: Time Management

Federation M.	Data Distribution M.
Declaration M.	Time Management
Object Management	Ownership M.

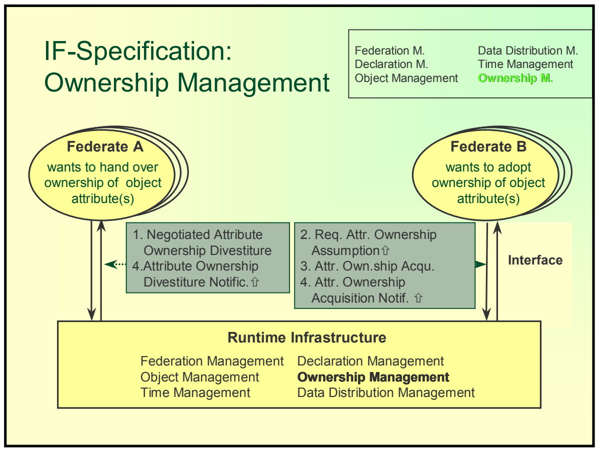
- “Lower Bound Time Stamp” of a federate:
  - if federate is time regulating: its current logical time + lookahead
  - if its non-time-regulating: positive infinity
- Query LBTS
  - Delivers current federation LBTS, i.e. the minimum of all LBTS’s of all participating federates
- Query Min Next Event Time
  - Minimum of all time-stamp ordered events that may be subsequently delivered to the federate
  - Takes into account messages which are still queued for delivery



IF-Specification: Ownership Management

Federation M.	Data Distribution M.
Declaration M.	Time Management
Object Management	**Ownership M.**

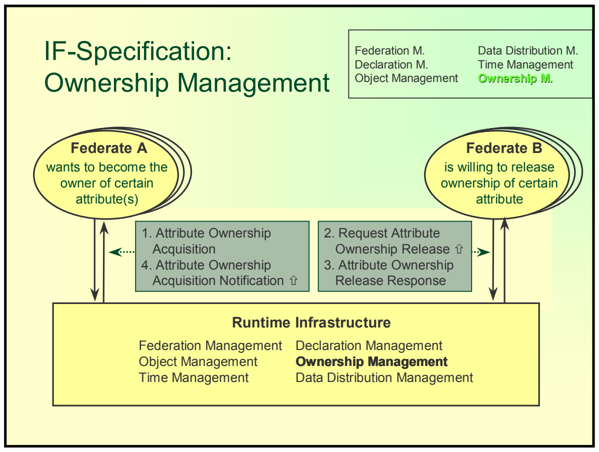
- Enables federates to transfer the ownership of object attributes
  - Ownership transfer based on federation execution design
  - Both push and pull mechanisms supported
- RTI services include:
  - Negotiated Attribute Ownership Divestiture / Attribute Ownership Acquisition
    - Federate wants to "get rid" of an attribute (push)
  - Attribute Ownership Acquisition / Attribute Ownership Release Response
    - Federate wants to become owner of an attribute (pull)
- Major design flaw: services are NOT time managed



IF-Specification: Ownership Management

Federation M.	Data Distribution M.
Declaration M.	Time Management
Object Management	**Ownership M.**

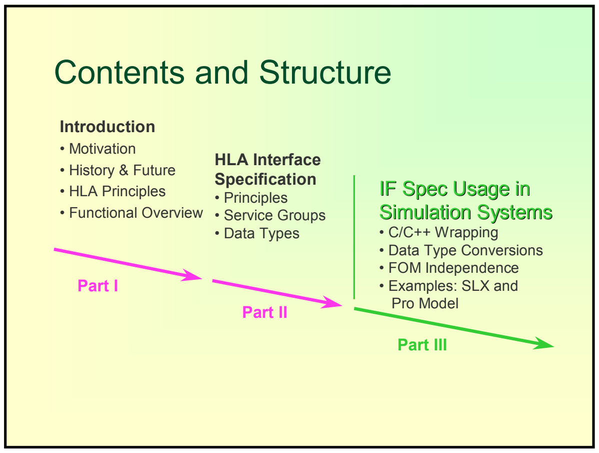
Federate A
wants to hand over ownership of object attribute(s)
Federate B
wants to adopt ownership of object attribute(s)
1. Negotiated Attribute Ownership Divestiture
4.Attribute Ownership Divestiture Notific. ⇧
2. Req. Attr. Ownership Assumption ⇧
3. Attr. Own.ship Acqu.
4. Attr. Ownership Acquisition Notif. ⇧
Interface
Runtime Infrastructure
Federation Management
Object Management
Time Management
Declaration Management
Ownership Management
Data Distribution Management



IF-Specification: Ownership Management

Federation M.	Data Distribution M.
Declaration M.	Time Management
Object Management	**Ownership M.**

Federate A
wants to become the owner of certain attribute(s)
Federate B
is willing to release ownership of certain attribute
1. Attribute Ownership Acquisition
4. Attribute Ownership Acquisition Notification ⇧
2. Request Attribute Ownership Release ⇧
3. Attribute Ownership Release Response
Runtime Infrastructure
Federation Management
Object Management
Time Management
Declaration Management
Ownership Management
Data Distribution Management



Contents and Structure

Introduction
- Motivation
- History & Future
- HLA Principles
- Functional Overview

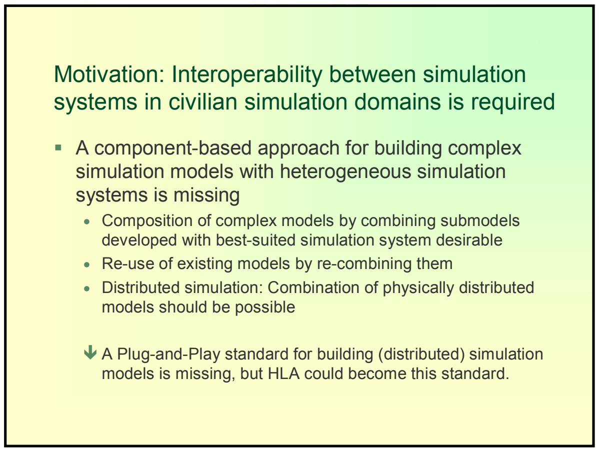
Part I

HLA Interface Specification
- Principles
- Service Groups
- Data Types

Part II

IF Spec Usage in Simulation Systems
- C/C++ Wrapping
- Data Type Conversions
- FOM Independence
- Examples: SLX and Pro Model

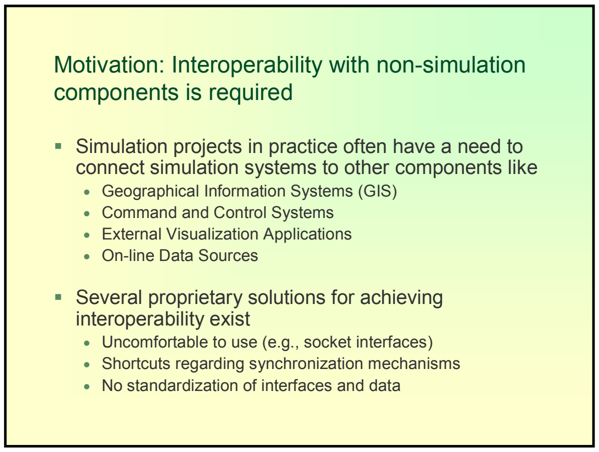
Part III



Motivation: Interoperability between simulation systems in civilian simulation domains is required

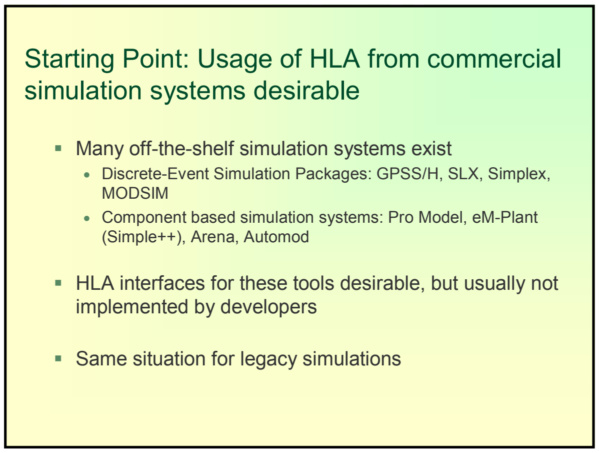
- A component-based approach for building complex simulation models with heterogeneous simulation systems is missing
  - Composition of complex models by combining submodels developed with best-suited simulation system desirable
  - Re-use of existing models by re-combining them
  - Distributed simulation: Combination of physically distributed models should be possible

  ↓ A Plug-and-Play standard for building (distributed) simulation models is missing, but HLA could become this standard.



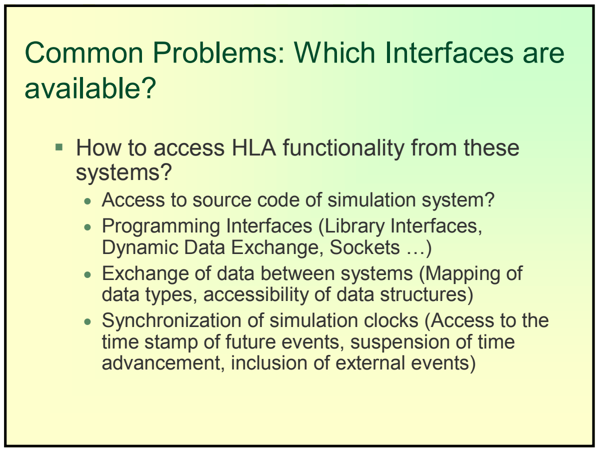
Motivation: Interoperability with non-simulation components is required

- Simulation projects in practice often have a need to connect simulation systems to other components like
  - Geographical Information Systems (GIS)
  - Command and Control Systems
  - External Visualization Applications
  - On-line Data Sources
- Several proprietary solutions for achieving interoperability exist
  - Uncomfortable to use (e.g., socket interfaces)
  - Shortcuts regarding synchronization mechanisms
  - No standardization of interfaces and data



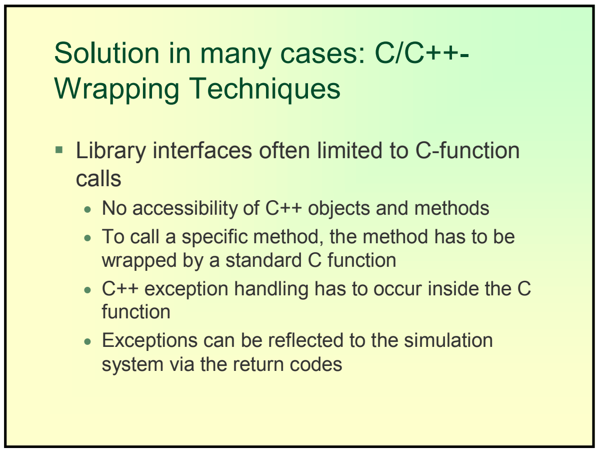
Starting Point: Usage of HLA from commercial simulation systems desirable

- Many off-the-shelf simulation systems exist
  - Discrete-Event Simulation Packages: GPSS/H, SLX, Simplex, MODSIM
  - Component based simulation systems: Pro Model, eM-Plant (Simple++), Arena, Automod
- HLA interfaces for these tools desirable, but usually not implemented by developers
- Same situation for legacy simulations



Common Problems: Which Interfaces are available?

- How to access HLA functionality from these systems?
  - Access to source code of simulation system?
  - Programming Interfaces (Library Interfaces, Dynamic Data Exchange, Sockets …)
  - Exchange of data between systems (Mapping of data types, accessibility of data structures)
  - Synchronization of simulation clocks (Access to the time stamp of future events, suspension of time advancement, inclusion of external events)



Solution in many cases: C/C++-Wrapping Techniques

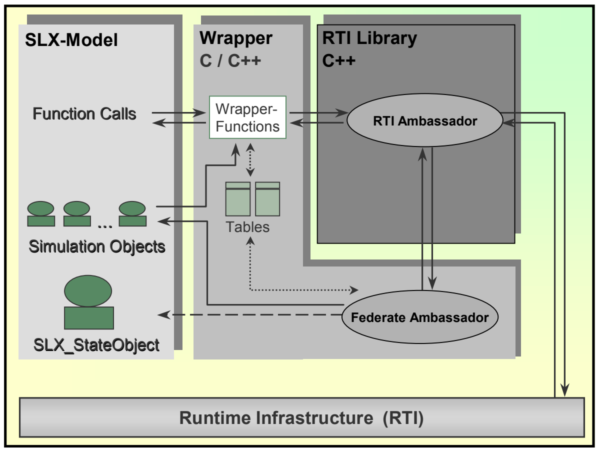
- Library interfaces often limited to C-function calls
  - No accessibility of C++ objects and methods
  - To call a specific method, the method has to be wrapped by a standard C function
  - C++ exception handling has to occur inside the C function
  - Exceptions can be reflected to the simulation system via the return codes

Sample Solution for SLX (1)

- SLX is a layered modeling system for discrete event simulation with powerful extensibility mechanisms
  - Statements concept
  - DLL interface
- Available for Windows 95/98/NT
- C-like syntax with selected concepts for object-oriented programming
- Developed by Wolverine Software

Sample Solution for SLX (2)

- Cannot define callback functions inside the SLX model
- Cannot call C++ methods directly
- Data-Types differ between SLX and standard C/C++
- Excellent simulation environment
- Can generate .H files
- Run-Time symbol table interrogation



Wrapper Function Examples (RTI1.0)

- Declaration in SLX

```
procedure RTI_RequestPause( string(*) PauseLabel )
returning boolean dll="slxrti10";
```

- Corresponding C-Function

```
int RTI_RequestPause( struct string_header* SLX_PauseLabel )
{
    try {
        rtiAmb->requestPause( SLX_PauseLabel->string_address);
    }
    catch ( RTI::Exception& e )
    {
        return FALSE;
    }
    return TRUE;
}
```

NextEvent-Request

(RTI1.0)

Request Time Advancement

Wait for Time Advance Grant

```
//Global Variables
bool     timeAdvGrant;
double  grantTime;

double RTI_NextEventRequest (double NextEventTime)
{
    try
    {
            timeAdvGrant = RTI::RTI_FALSE;
            ms_rtiAmb->nextEventRequest(NextEventTime);
    }
    catch ( RTI::Exception& e )
    {
            return (-1);  //simplified
    }
    while (timeAdvGrant == RTI::RTI_FALSE)
    {
            int eventsToProcess = 1;
            while ( eventsToProcess )
            {
                        eventsToProcess = ms_rtiAmb->tick();
            }
    }
    return (grantTime);
}
```

Data Type Conversion between simulation system and RTI often necessary

- Representation of simulation time: most tools use double values, RTI uses a separate class

```
class RTI_EXPORT_FEDTIME RTIfedTime :

public RTI::FedTime {
// Constructors and Destructors
public:         RTIfedTime();
                RTIfedTime(const RTI::Double&);
                RTIfedTime(const RTI::FedTime&);
                ...
                virtual ~RTIfedTime();

// Implementation functions
public:         virtual RTI::Double getTime() const;
                ...
```

NextEvent-Request

(RTI1.3)

Build instance of Time Class from double value

Fetch double value from class

```
//Global Variables
bool            timeAdvGrant;
RTIfedTime      grantTime;

double RTI_NextEventRequest (double NextEventTime)
{
   try
   {
         timeAdvGrant = RTI::RTI_FALSE;
         ms_rtiAmb->nextEventRequest(
                               ( RTIfedTime (NextEventTime));
   }
   catch ( RTI::Exception& e )
   {
         return (-1);  //simplified
   }
   while (timeAdvGrant == RTI::RTI_FALSE)
   {
         int eventsToProcess = 1;
         while ( eventsToProcess )
         {
                  eventsToProcess = ms_rtiAmb->tick();
         }
   }
   return ((double) grantTime.getTime());
}
```

Transfer of data over system boundaries

- Simulation systems differ in their capabilities to access/modify internal data objects
  - SLX
    - Pointers to internal data objects can be passed via the library interface
    - External modifications of data objects can be reflected as events into the simulation
  - Pro Model
    - No access to data objects via the library interface (XSUB)
    - Data can only be received via the return value of an external function, data type is limited to double

Transfer of data over system boundaries – Example (1)

- Function of the SLX-HLA-IF
  - Registers an object with the RTI
  - Passes a pointer to the wrapper library which points to the actual SLX object
  - Returns a unique ID under which the object is known by the RTI and the wrapper

```
procedure RTI_RegisterObjectInstance(
     string(*)   ObjectClassName,
     pointer(*)  theObject)
returning int dll="slxrti13";
```

Transfer of data over system boundaries – Example (2)

- Function of the SLX-HLA-IF
  - Sends an update for the previously registered object
  - No attribute data is passed via the DLL interface !!!
  - Wrapper stores pointer passed in RTI_Register-Object and uses it to access attribute data directly

```
procedure RTI_UpdateAttributeValues(
    int         Object_ID,
    string(*)   AttributeList,
    double      TimeStamp)
returning int dll="slxrti13ng";
```

Mapping of data types to a specific FOM

- Simulation systems often define their own data types and with proprietary implementations
  - SLX
    - “double”, “float”: always 64 Bits
    - Only one “integer” type (no short/long etc.)
    - Special implementation of “string”
- Conversions to and from the data types mandates by a specific FOM are necessary

FOM-Independence/FOM-Agility

- Ability of software to adapt to different FOMs by defining mappings
  - A mapping at runtime between a general FOM class “TRUCK” and specific subtypes of TRUCK modeled by the simulation
- Ability of software to convert units of data
  - E.g., conversion of Kilometers to Miles
- Implementation of software which is independent from any FOM, e.g., for implementing general HLA interfaces for simulation systems
  - SLX-HLA-Interface has no knowledge at compile time about FOM contents
  - At runtime, the SLX model provides the required information

The Synchronization Issue (1)

- Simulation clocks typically need to be synchronized with other participants
- Most commonly, conservative synchronization will be appropriate
  - Easiest to implement
  - Tools are not capable to rollback/recover
- Event based conservative synchronization requires access to time stamp of next scheduled event
- Alternately time stepped conservative synchronization can be used

The Synchronization Issue (2)

- Add a special synchronization thread to the simulation model
- Acts as last event at a specific simulation time
- Determines time stamp of next scheduled event
- Requests the time advancement
- Receives zero or more external events
- Advances simulation time to time granted

SLX-Synchronization Thread

```
double grantTime;                          // stores the time returned from RTI
double nextEventTime;                      // stores the time stamp of the next event

forever
{
    nextEventTime = next_imminent_time();   // determine time stamp of next
                                            // internal event

    grantTime = RTI_NextEventRequest( nextEventTime);
                                            //request advancement

    wait until (time == grantTime);         // advance to grant time

    RTI_ReflectControlVariableChanges();    // Let external events take effect

    ...                                    // query and process any external events

    yield;                                 // hand over control to other simulation
                                             threads
}
```

Pro Model - Synchronization Thread

```
//Request advancement to next time step
GrantTime = XSUB(  ProModel_RTI13,
                   "RTI_TimeAdvanceRequest",
                   (CLOCK(sec)+1))

//Query for any attribute changes or received interactions
   ...

// Wait for the time step to elapse, then start from top
WAIT GrantTime - CLOCK(sec)
```

Summary

- HLA offers a solution for interoperability for general simulation systems
  - Interoperability between platforms, languages and time advancement mechanisms
- Initial effort for constructing a general HLA interface for a simulation system

Summary

- For many systems, an adopted HLA API is necessary
  - C/C++ wrapping often needed
  - Simplifications of the HLA API can be implemented along the way
  - Data type conversions
- Simulation system specific HLA interfaces can be implemented independent from FOMs