Simulation Interoperability Standards Organization

"Simulation Interoperability & Reuse through Standards"

News in HLA 4 RTI and API for Developers

2024-SIW-Presentation-19

Mikael Karlsson, Pitch Technologies, Sweden

HLA Evolved to HLA 4

This presentation provides a detailed look at the changes from HLA Evolved to HLA 4 from a developer perspective.

We look at changes in services, API changes, etc.

This presentation does not cover the history of simulation, policy making, etc. This is for federate (and RTI) developers.

RtiConfiguration

HLA Evolved had *Local Settings Designator (LSD)* – a string that primarily contained a *Configuration Name* which is a reference to a configuration.

But RTI implementations accepted other settings in LSD in an ad-hoc, implementation-specific format.

```
String LSD =
   ”MyConfiguration\n” +
   ”crcAddress=192.168.11.5:8989\n” +
   ”secretSetting=5”;
connect(myFederateAmbassador, HLA_IMMEDIATE, LSD);
```

RtiConfiguration

HLA 4 formalizes the configuration using a new **RtiConfiguration** object with the following fields:

- configurationName
- rtiAddress
- additionalSettings (still implementation-specific format)

```
config = RtiConfiguration.createConfiguration()
   .withConfigurationName(”MyConfiguration”)
   .withRtiAddress(”192.168.11.5:8989”)
   .withAdditionalSettings(”secretSetting=5”);
connect(myFederateAmbassador, HLA_IMMEDIATE, config);
```

Federate Authorization

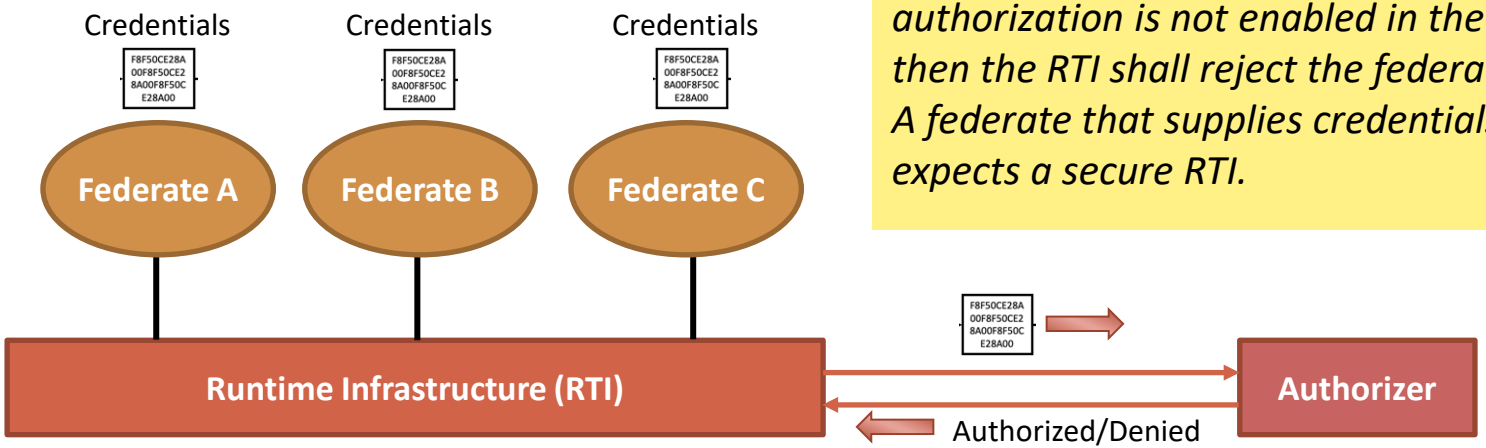
- The RTI can require authorization for certain operations (see list on the right).
- Federate provides credentials in connect call.
- When federate attempts an operation that requires authorization, the RTI checks the federate's credentials to approve or reject the operation.

Authorized operations
- connect
- createFederationExecution
- destroyFederationExecution
- listFederationExecutions
- joinFederationExecution

```
myCredentials = new HLAplainTextPassword(”VerySecret”)
connect(myCredentials)
joinFederationExecution(”PrivateFederation”) => Unauthorized
```

Authorization

- To perform authorization, the RTI passes the federate's credentials to Authorizer that approves or rejects the operation.



If a federate supplies credentials, but authorization is not enabled in the RTI, then the RTI shall reject the federate. A federate that supplies credentials expects a secure RTI.

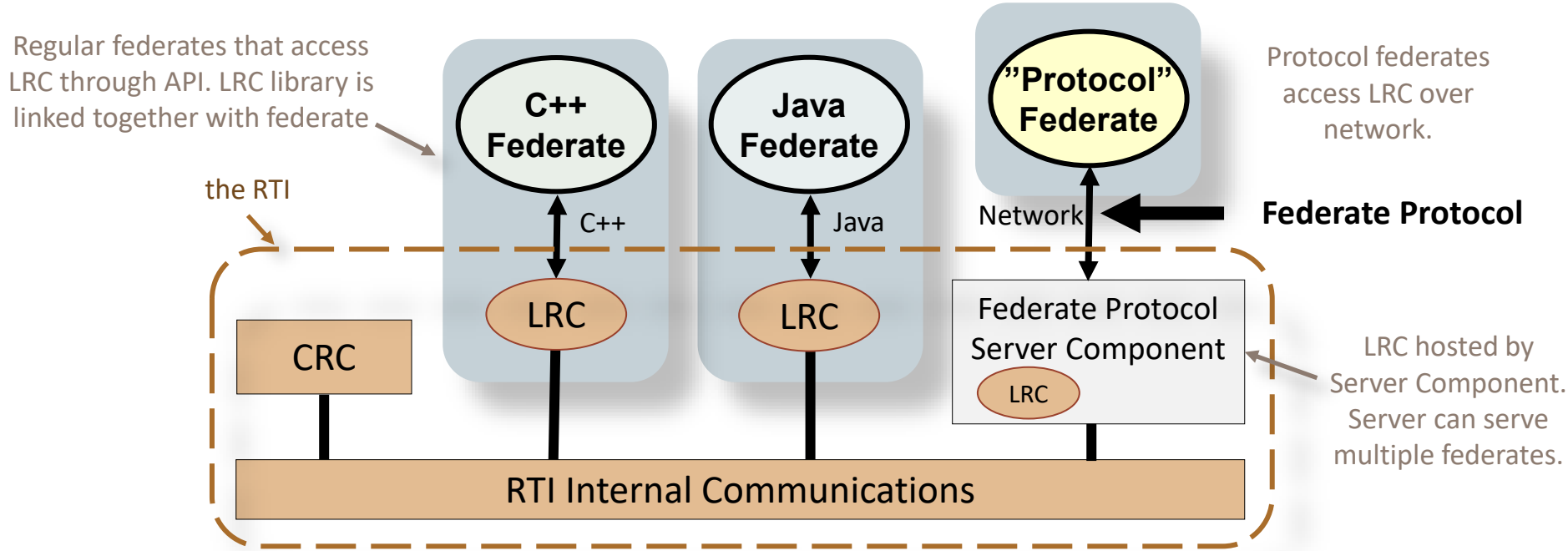
Authorization

- The Authorizer is a plugin. The choice of authorizer is configured in the RTI.

- All RTIs shall be delivered with a built-in authorizer called *HLAauthorizer* that shall support plaintext passwords. It may support other types of credentials.

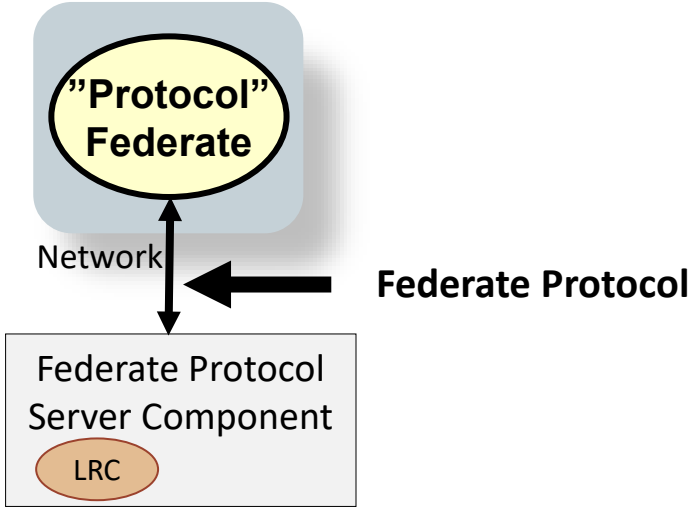
- Custom authorizer plugin may support other types of credentials, e.g. certificates or Active Directory.

Federate Protocol

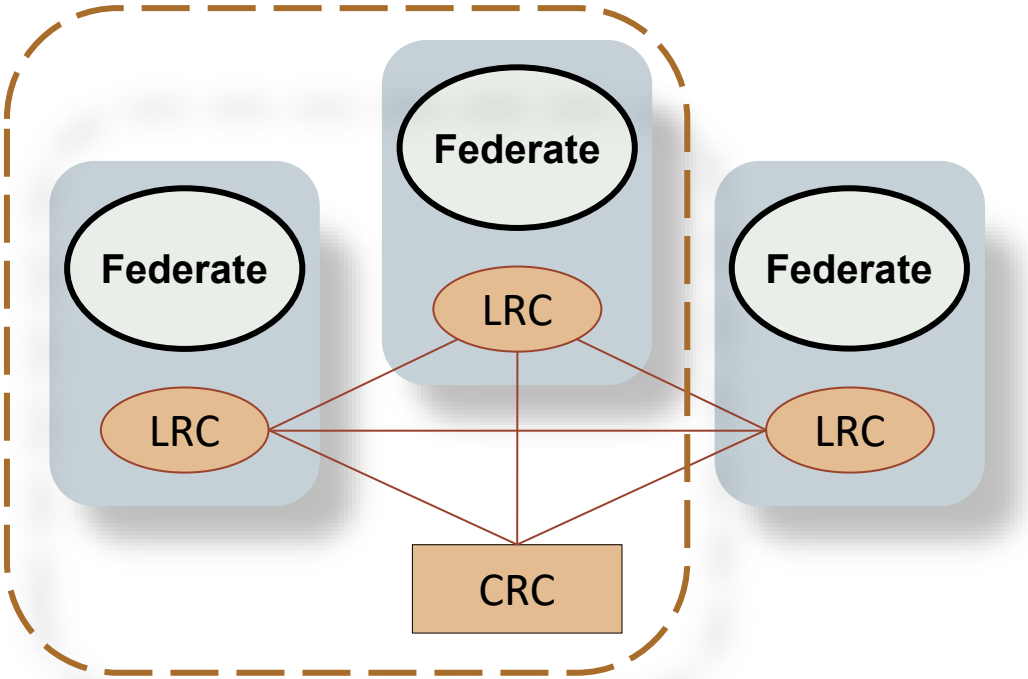


Federate Protocol Benefits

- A documented protocol.
  - Easier security review
  - WebSocket is easier to get through gateways, etc.
- Separates federate from RTI library.
  - Additional languages/OS
  - Reduced CPU load and network traffic
  - RTI can be upgraded without touching federate installation.
  - Simplifies accreditation
- Cloud friendly
  - Federate has a single connection to RTI Server.
  - All LRCs live in the cloud for easy peer-to-peer connectivity.

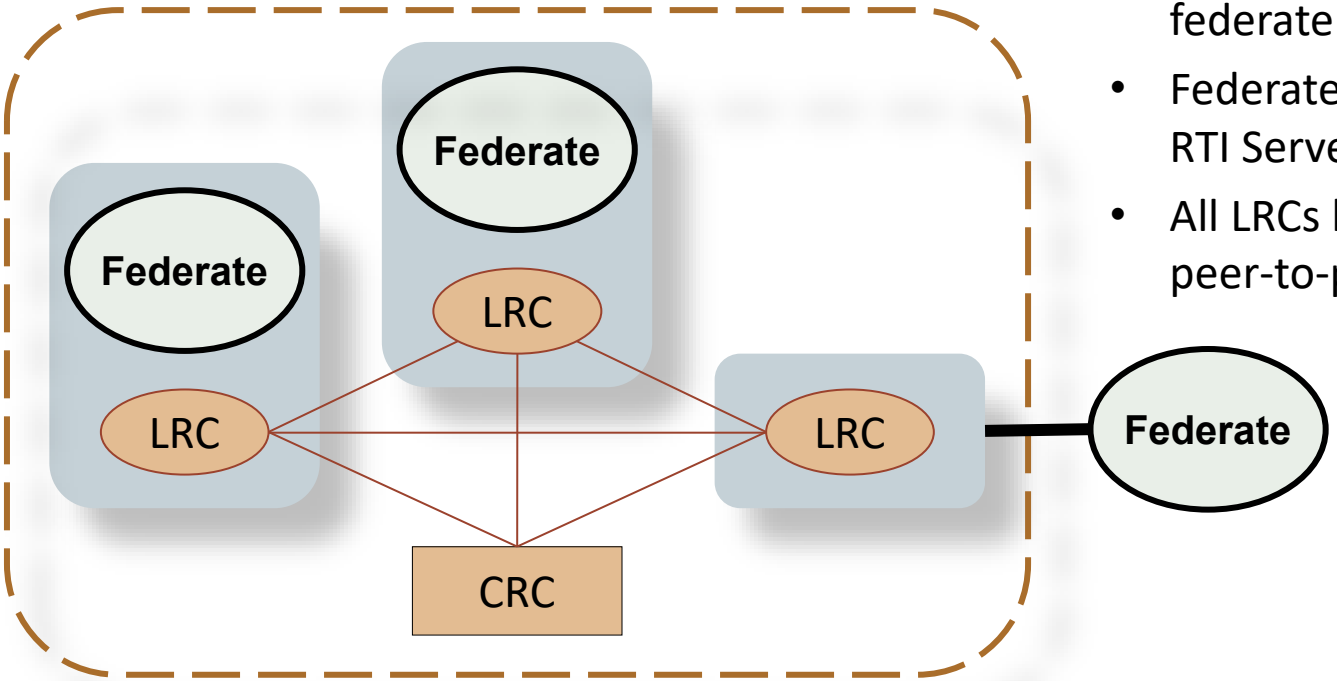


Federate Protocol Benefits



- We have a federation.
- We want to put part of it in the cloud.
- This creates issues due to the peer-to-peer connections between LRCs.

Federate Protocol Benefits



- Using Federate Protocol, we put the LRC in the cloud and the federate outside.
- Federate has a single connection to RTI Server.
- All LRCs live in the cloud for easy peer-to-peer connectivity.

Protocol Buffers

The Federate Protocol network format is defined using Protocol Buffers (protobuf).

Protocol Buffers is a free and open-source cross-platform data format used to serialize structured data.

```
message InteractionClassHandle {
    bytes data = 1;
}

message GetInteractionClassHandleRequest {
      string interactionClassName = 1;
}

message GetInteractionClassHandleResponse {
    InteractionClassHandle result = 1;
}
```

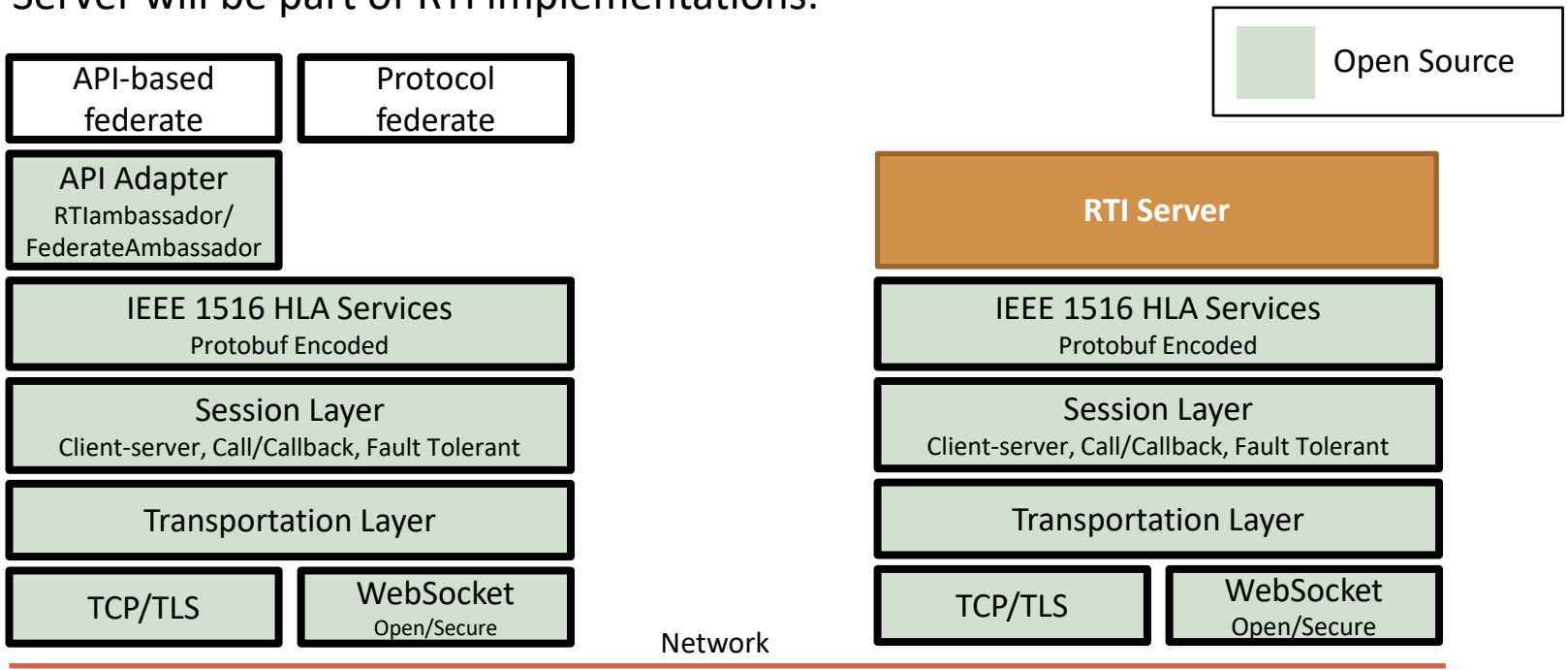
Protocol description is compiled using protoc compiler into source code that is used to build and parse messages into byte arrays.

Code can be generated in many languages, e.g. Java, C++, Python, Objective-C, C#, Ruby, Go, JavaScript, etc.

Federate Protocol Stack

Lower level components and an API adapter (Java and C++) will be Open Source.

RTI Server will be part of RTI implementations.



Raw protobuf usage by ”Protocol” Federate

```
_transport = TransportFactory.createTcpTransport("localhost")
_clientSession = SessionFactory.createSession(_transport);
_clientSession.start(this);

AtomicReference<InteractionClassHandle> messageClass = new AtomicReference<>();
GetInteractionClassHandleRequest getInteractionClassHandleRequest = GetInteractionClassHandleRequest.newBuilder()
    .setInteractionClassName("Communication")
    .build();
CallRequest callRequest = CallRequest.newBuilder()
    .setGetInteractionClassHandleRequest(getInteractionClassHandleRequest)
    .build();
CompletableFuture<byte[]> futureResponse = _clientSession.sendHlaCallRequest(callRequest.toByteArray());
futureResponse.thenAccept(bytes -> {
  try {
    CallResponse callResponse = CallResponse.parseFrom(bytes);
    GetInteractionClassHandleResponse response = callResponse.getGetInteractionClassHandleResponse();
    messageClass.set(response.getResult());
  } catch (InvalidProtocolBufferException e) {
    throw new RuntimeException("Failed to parse response.");
  }
});
```

Using the Federate Protocol API adapter

Java

Replace RTI jar on classpath with API adapter jar.

Done!

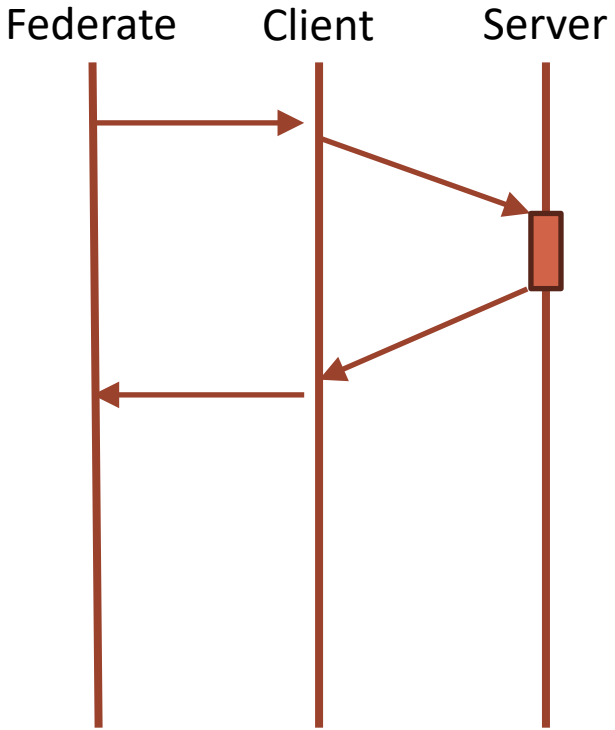
C++

Replace RTI DLL with API adapter DLL.

Done!

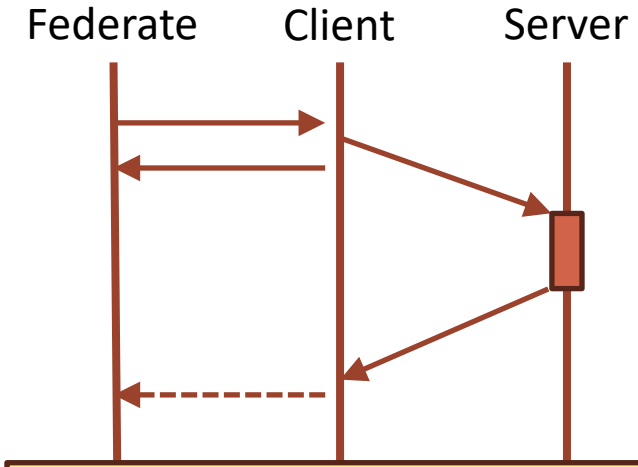
Synchronous calls and latency

- Standard HLA API is synchronous. Calls do not return until they have received a result from the RTI. Even methods that have no return value have to wait for success/failure.
- Long latency affects performance.



Async API

- The API adapter includes a custom asynchronous API variant which is **not** part of the standard.
- The call to the client returns immediately.
- The call returns a Future where the return value is filled in when it arrives from the server.



The API adapter has a flag that makes some synchronous updates and interactions asynchronous by simply ignoring the success/failure response.

Async API

The asynchronous methods return a **Future** – a construct that lets the caller retrieve the result at a later time.

```
Future<ObjectClassHandle> futureA =
      _rtiAmbassador.async().getObjectClassHandle(”A”);
Future<ObjectClassHandle> futureB =
      _rtiAmbassador.async().getObjectClassHandle(”B”);

// Do something else
...

// Retrieve the values
ObjectClassHandle clsA = futureA.get();
ObjectClassHandle clsB = futureB.get();
```

Relaxed Advisories

- In HLA Evolved, advisories are based on instance-specific data (known class). This scales badly.

- In HLA 4, advisories are based on subscription data which is much more scalable.
- You don’t have to change anything in your federate to take advantage of this. Just change a switch in the FOM.

Relaxed DDM

- In HLA Evolved, DDM filtering had to be perfect, no unwanted updates may leak through. This prevents some optimized solutions.

- In HLA 4, Relaxed DDM allows DDM filtering to be Best Effort, i.e. unwanted updates are allowed to leak through.
- Your federate must be prepared to accept some updates that have non-overlapping regions. If you're using DDM to reduce network traffic then this will not matter.

Simplified Callbacks

In HLA Evolved, callbacks had many different overloads.

- `reflectAttributeValues(theObject, theAttributes, userSuppliedTag, sentOrdering, theTransport, reflectInfo)`
- `reflectAttributeValues(theObject, theAttributes, userSuppliedTag, sentOrdering, theTransport, theTime, receivedOrdering, reflectInfo)`
- `reflectAttributeValues(theObject, theAttributes, userSuppliedTag, sentOrdering, theTransport, theTime, receivedOrdering, retractionHandle, reflectInfo)`

HLA 4 cleans up and reduces number of overloads

- `reflectAttributeValues(objectInstance, attributeValues, userSuppliedTag, transportationType, producingFederate, optionalSentRegions)`
- `reflectAttributeValues(objectInstance, attributeValues, userSuppliedTag, transportationType, producingFederate, optionalSentRegions, time, sentOrderType, receivedOrderType, optionalRetraction)`

Optional parameters are null if unavailable.

User-supplied Tag on more Ownership Operations

A number of ownership related calls and callback gain a user-supplied tag argument

`unconditionalAttributeOwnershipDivestiture`

`attributeOwnershipAcquisitionIfAvailable`

`attributeOwnershipReleaseDenied`

`attributeOwnershipDivestitureIfWanted`

`requestDivestitureConfirmation`

`attributeOwnershipUnavailable`

This gives more opportunities to track and control ownership operations.

Change Default Transportation/Order Type

- When the federate acquires ownership of an attribute, through registration or ownership transfer, the attribute gets the default order type.
- Initial value for default order type comes from FOM.
- `changeDefaultAttributeOrderType` changes the default order type for an attribute on the calling federate.
- The corresponding functionality is available for Transportation Type.

FOM references

- In HLA Evolved, the Java API expects FOM references as URLs.
- In C++, FOM references were strings, typically filenames.

- In HLA 4, the behavior is unified.
- Both Java and C++ accepts FOM references as strings.
- String can be URL or filename.

Switches

- In HLA Evolved, switches are controlled through various ways. Some are controlled in the FOM, some using API calls, and yet others through MOM.

- In HLA 4, switch management is unified.
- All switches can be set in FOM.
- API has set and get methods for all switches.
- All switches are available in MOM.

Miscellaneous

- New callback: `federateResigned(reasonForResign)`
- New service: `listFederationExecutionMembers`
- Producing Federate callback argument is mandatory and always delivered.
- New callback: `flushQueueGrant`. Matches `flushQueueRequest`.
- `normalizeObjectClassHandle`
- `normalizeInteractionClassHandle`
- `normalizeObjectInstanceHandle`

C++11 Compatible API

- HLA Evolved C++ API uses constructs that have been deprecated in C++11.
- `std::auto_ptr` -> `std::unique_ptr`
- Exception specifications removed
- (Actually, the standard says C++14 which is basically C++11 plus some bugfixes)

C++ Library names

- In HLA Evolved, C++ libraries built using different compilers all shared the same name. This led to a lot of PATH juggling.

- In HLA 4, C++ libraries have names that show which compiler they were built with, e.g. gcc73 or vc142.
- This means that all libraries can live in the same directory. No need to modify PATH based on compiler.

RTI Summary

- RtiConfiguration
- Federate Authorization
- Federate Protocol
- Relaxed Advisories
- Relaxed DDM
- Simplified Callbacks
- User-supplied tags on Ownership operations
- Change Default Attribute Order Type
- Change Default Attribute Transportation Type
- FOM reference is string
- Simplified switches
- C++11 compatible API
- C++ Library names

Open Source Federate Protocol Client

- The Open Source Federate Protocol Client is now available.
- https://github.com/Pitch-Technologies/FedProClient

(Slide based on “Blank”)

Simulation Interoperability Standards Organization

"Simulation Interoperability & Reuse through Standards"

Q&A / Discussion