Simulation Interoperability Standards Organization

"Simulation Interoperability & Reuse through Standards"

Deep dive into the HLA 4 Federate Protocol

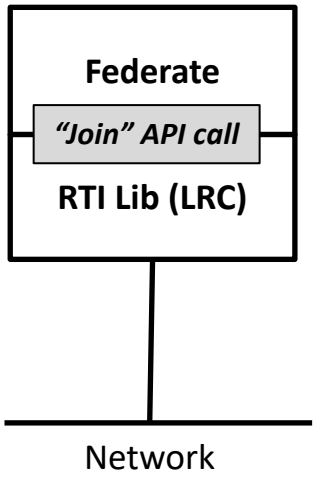
SIM-2023-Presentation-04

Fredrik Antelius, Pitch Technologies, Sweden

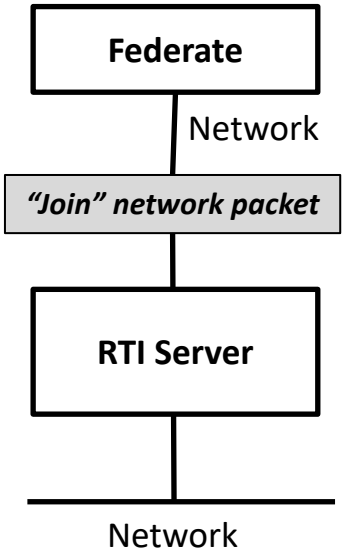
HLA Federate Protocol

- Federates can call the RTI using a protocol instead of calling a local library
- All HLA services are available
- Supports all FOMs
- Can be mixed with traditional federate deployment

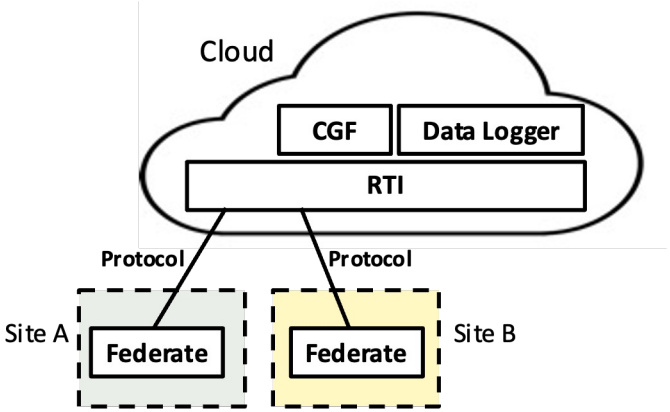
Standardized API

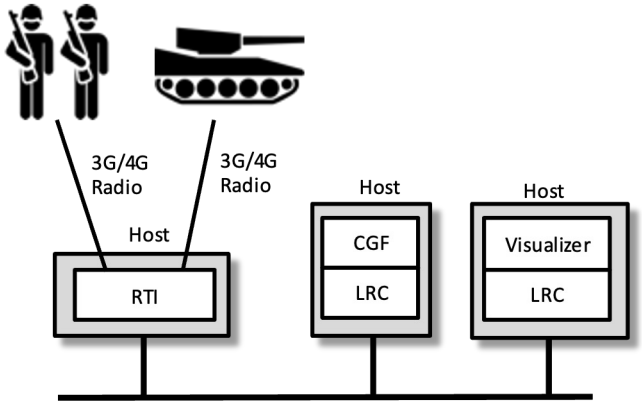


Standardized protocol



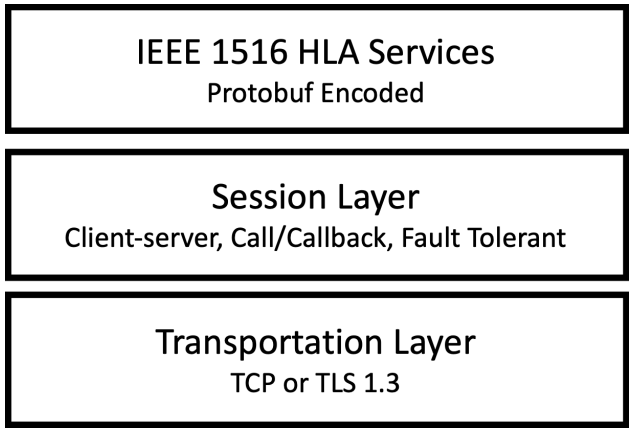
Use Cases





- **HLA Federate Protocol communicates like most Internet applications do**
  - Mix federates in cloud and on-premises
- **Better support for live deployment**
  - Fault tolerance and recovery when using unreliable links
  - Point-to-point communication more suitable for 3G/4G/5G
- **Support for more programming languages**
  - Dynamically linking C++ or Java libraries can be problematic

HLA Federate Protocol Layers



- IEEE 1516 HLA Services Layer with HLA service calls and callbacks
- Session Layer supports session setup and teardown, HLA calls & callbacks and fault tolerance
- Transportation Layer based on TCP/IP or secure TLS 1.3

Transport Layer

- **Uses a regular TCP connection**
  - Stream of binary data
  - Point-to-point connection
- **Federate initiate the connection (act as client)**
- **RTI server listens for incoming connections**
  - Several federates can connect to the same RTI server
- **Federate can connect from anywhere**
- **Firewall and container friendly**
  - No UDP or multicast
  - No incoming connections to federate host

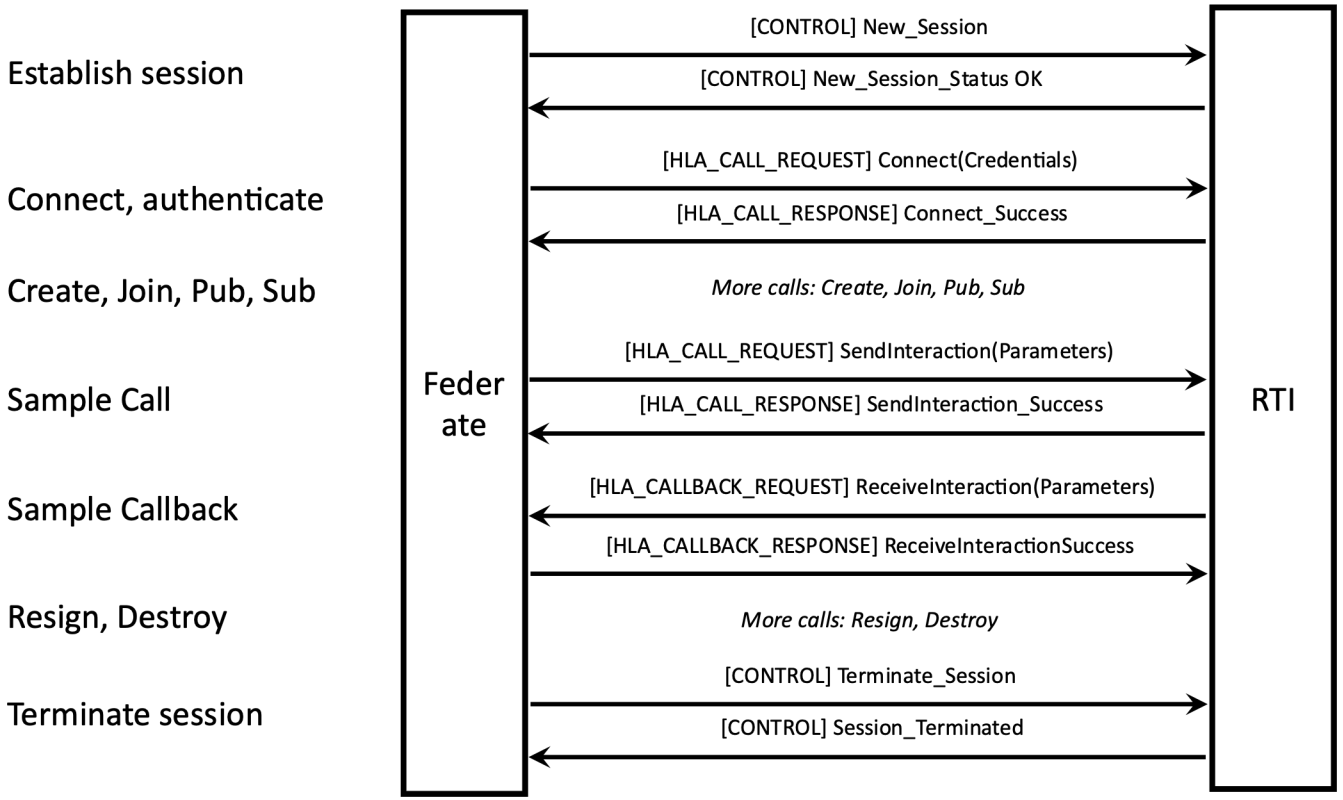
Client Transport Layer in Kotlin

```kotlin
class Transport {
    private lateinit var _socket: Socket

    fun connect(host: String, port: Int) {
        _socket = try {
            Socket(host, port)
        } catch (e: IOException) {
            throw RtiException("Failed to connect to RTI Server at '$host:$port'", e)
        }
    }

    fun writeMessage(message: Message) {
        with(_socket.getOutputStream()) { this: OutputStream!
            write(message.encode())
            flush()
        }
    }

    fun readMessage(): Message {
        return Message.decode(_socket.getInputStream())
    }
}
```

Session Layer – Messages

- **12 Messages defined in Session layer**
  - Pair of Request–Response messages
  - Federate sends a request, then the RTI sends result back in a response
    - *RTI can send requests for HLA callbacks*
- **Control Messages**
  - New Session request & response
  - Terminate Session request & response
- **HLA Messages**
  - HLA Call request & response
  - HLA Callback request & response

Message Exchange – Example



Sending – Client Session Layer in Kotlin

```kotlin
private fun sendTerminateSession() {
    _transport.writeMessage(
        Message.with(_sequenceNumber++, _sessionId, Message.Type.CTRL_TERMINATE_SESSION, NO_PAYLOAD)
    )
}

fun sendHlaCallbackResponse(hlaCallbackResponse: ByteArray) {
    _transport.writeMessage(
        Message.with(_sequenceNumber++, _sessionId, Message.Type.HLA_CALLBACK_RESPONSE, hlaCallbackResponse)
    )
}

fun sendHlaCallRequest(hlaCallRequest: ByteArray): CompletableFuture<ByteArray> {
    val sequenceNumber = _sequenceNumber++
    _transport.writeMessage(
        Message.with(sequenceNumber, _sessionId, Message.Type.HLA_CALL_REQUEST, hlaCallRequest)
    )
    val future = CompletableFuture<ByteArray>()
    _hlaCallsInProgress[sequenceNumber] = future
    return future
}
```

Receiving – Client Session Layer in Kotlin

```kotlin
private fun runMessageReaderLoop() {
    while (true) {
        val message = _transport.readMessage()
        assert(message.sessionId == _sessionId)
        when (message.type) {
            Message.Type.HLA_CALL_RESPONSE → handleHlaCallResponse(message)
            Message.Type.HLA_CALLBACK_REQUEST → handleHlaCallbackRequest(message)

            Message.Type.CTRL_SESSION_TERMINATED →
                // done, termination complete
                return

            else → assert(false)
        }
    }
}
```

HLA Service Layer

- **RTI Ambassador Services**
  - HLA Call request & response
- **Federate Ambassador Services**
  - HLA Callback request & response
- **HLA Calls and Callbacks are independent**
  - Multiple ongoing requests
- **Binary data encoded using Protobuf**
  - Encoder and decoders are generated

```
message SendInteractionRequest {
      InteractionClassHandle theInteraction = 1;
      ParameterHandleValueMap theParameters = 2;
      bytes userSuppliedTag = 3;
}

message SendInteractionResponse {
}


message CallResponse {
  oneof callResponse {
    ExceptionData exceptionData = 1;
    ConnectResponse connectResponse = 2;
    ConnectWithCredentialsResponse connectWithCred
    ConnectWithConfigurationResponse connectWithCo
    ConnectWithConfigurationAndCredentialsResponse
    DisconnectResponse disconnectResponse = 6;
```

Send Interaction – Client HLA Service Layer in Kotlin

```kotlin
fun sendInteraction(
    interactionHandle: InteractionClassHandle,
    parameters: Map<ParameterHandle, ByteArray>,
) {
    val callRequest = CallRequest(
        sendInteractionRequest =
        SendInteractionRequest(interactionHandle, convert(parameters), ByteString.EMPTY)
    )
    val response = sendHlaCallRequest(callRequest.encode())
    val result = decodeCallResponse(response.join())
    throwOnException(result)
    assert(result.sendInteractionResponse ≠ null)
}
```

Receive Interaction – Client HLA Service Layer in Kotlin

```kotlin
fun dispatchHlaCallback(sequenceNumber: Int, hlaCallback: ByteArray) {
    val callbackRequest = decodeCallbackRequest(hlaCallback)
    if (callbackRequest.receiveInteraction ≠ null) {
        val interactionHandle = callbackRequest.receiveInteraction.interactionClass!!
        val parameters = convert(callbackRequest.receiveInteraction.theParameters!!.parameterHandleValue)

        _federateAmbassador.receiveInteraction(interactionHandle, parameters)
    } else if (callbackRequest.objectInstanceNameReservationSucceeded ≠ null) {
        val objectName = callbackRequest.objectInstanceNameReservationSucceeded.objectName

        _federateAmbassador.objectInstanceNameReservationSucceeded(objectName)
    } else if (callbackRequest.objectInstanceNameReservationFailed ≠ null) {
```

Transport Layer details

- **Provides back-pressure and flow control**
  - Bundling may be performed
- **TCP transport must be supported**
- **Option to use TLS 1.3 for encrypted and authenticated communication**
  1. Encrypted Mode only
  2. Server Authenticated Mode
     - *Federate can verify the RTI identity via certificate trust chain*
     - *Like*  *in a web browser*
  - Can be combined with HLA 4 Authorized Federate
- **Option to use WebSocket as transport**
  - WebSocket is available in the browser
  - Easy integration with firewalls and HTTP proxies for load balancing, routing and TLS

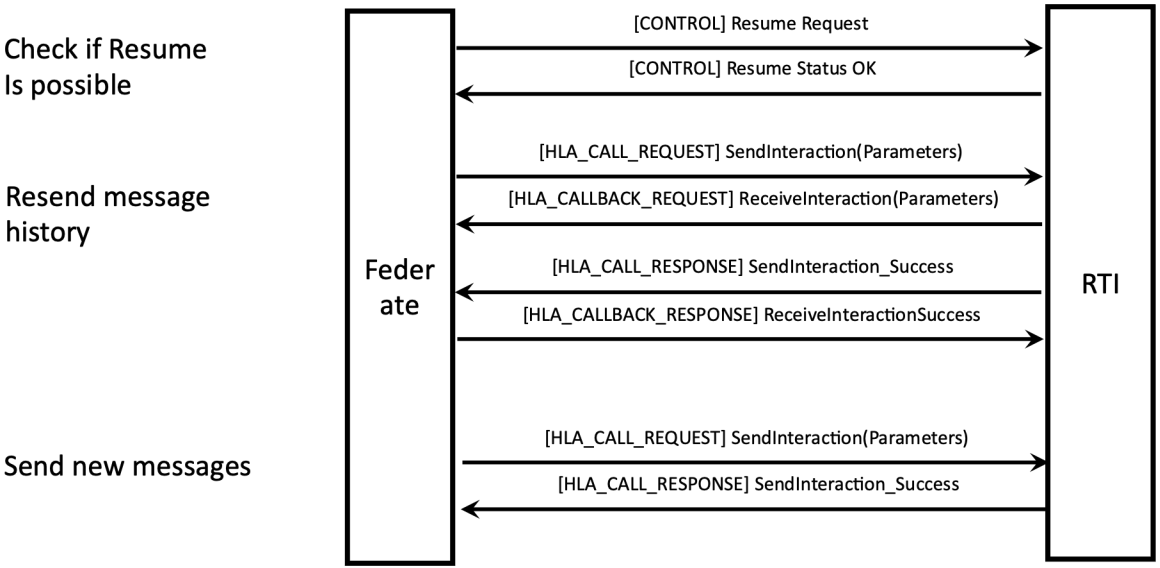
Session Layer details

- **Each session has a unique Session Id**
  - Assigned by RTI when a new session is initiated
- **Each message has a unique Sequence Number**
  - Response message needs to be matched with the corresponding request
  - Response messages contains a “in response to” Sequence Number
- **Packet size is included in header**
  - No need for framing in the transport layer
- **Heartbeats are sent regularly**
  - Federates are expected to send a heartbeat every 60 s and the RTI should respond
  - To detect any connectivity issues
  - To keep the connection open through firewalls and proxies
- **Limit number of ongoing requests**
  - To not overwhelm RTI or Federate
  - No hard limit in the standard

Fault Tolerant Session Layer

- **A broken connection can be resumed**
  - The transport layer connection can be interrupted and reconnected
- **Control Messages to resume a session**
- **Session Layer keeps a Message History Buffer**
  - Sent HLA messages are stored in a message history buffer
  - The history buffer has a limited size
    - *Last Received Message field in header can be used to prune history buffer*
- **If the transport can be reconnected, the session may be resumed**
  - During resume, the federate and RTI exchange:
    - *Sequence number of last received message*
    - *Sequence number of oldest message in history buffer*
  - If all messages that have been lost are in the history buffer, then the lost messages are resent, and the session is resumed
  - If the history buffer does not contain all the lost messages, then the resume fails
- **Resume will fail if the connection can not be reestablished fast enough**

Resume Example



1. **The federate's first message is a Resume Request**
   - *Sequence number of Last received message from RTI and oldest message in history buffer*
2. **RTI responds with Resume Status OK**
   - *After verifying that all necessary messages are in Federate and RTI history buffers*
3. **Lost messages are resent from history buffers**
4. **Session is restored and new messages may be exchanged**

HLA Services Layer details

- **Callback model is always Immediate**
  - Evoke and Enable/Disable Callbacks HLA services are not available
- **Protobuf version 3**
  - Generate encoders and decoders from message specification
  - Support for many programming languages
  - Initially developed by Google, other implementations are available
- **Encoding of HLA service parameters in protobuf**
  - Handles are represented using regular encoded binary format
  - Logical time, timestamps and lookahead are encoded to binary data
  - FOM is binary file content, zip file or URL
  - Multiple protobuf messages to handle optional arguments to HLA services

Client API for Federate Protocol?

- **Federate Protocol is a network protocol, not an API**
- **How does a federate use the Federate Protocol?**
  - Directly, by reading and writing network messages
  - Create idiomatic API for languages like Kotlin, Rust, JavaScript or Go
  - Create tailored API for a specific use case
  - Reuse HLA C++ and Java API
    - *Easy to switch between Federate Protocol and traditional mode*
- **Federate Protocol via the standard HLA API**
  - Asynchronous protocol with multiple ongoing requests
  - Synchronous API with return values and exceptions
    - *Supports only **one** ongoing request!*
    - ***Very sensitive to roundtrip latency** between federate and RTI!*
  - Ignore exceptions for ”safe” services like Update Attribute Value and Send Interaction?
  - Create asynchronous API using Promise and Future?

Send Interaction – Federate in Kotlin

```
println("Type messages you want to send. To exit, type . <ENTER>")
while (true) {
    print("> ")
    val message = reader.readLine()
    if (message == ".") {
        break
    }

    _rtiAmbassador.sendInteraction(
        _communicationClassHandle, mapOf(
            _messageParameterHandle to HlaUnicodeString.encode(message),
            _senderParameterHandle to HlaUnicodeString.encode(_userName)
        )
    )
}

_rtiAmbassador.resignFederationExecution()
_rtiAmbassador.destroyFederationExecution(FEDERATION_NAME)
_rtiAmbassador.disconnect()
```

Receive Interaction – Federate in Kotlin

```kotlin
override fun receiveInteraction(
    interactionHandle: InteractionClassHandle,
    parameters: Map<ParameterHandle, ByteArray>
) {
    if (interactionHandle == _communicationClassHandle &&
        parameters.containsKey(_messageParameterHandle) &&
        parameters.containsKey(_senderParameterHandle)
    ) {
        val message = HlaUnicodeString.decode(parameters[_messageParameterHandle]!!)
        val sender = HlaUnicodeString.decode(parameters[_senderParameterHandle]!!)
        println("$sender: $message")
        print("> ")
    }
}
```

Client library as Open-Source Software

- **Chat federate with Federate Protocol client in Kotlin**
  - No error handling and no support for resume
  - github.com/Pitch-Technologies/KotlinChat
- **Federate Protocol Client library with HLA C++ and Java API**
  - Coming soon!
  - Help welcome!
  - github.com/Pitch-Technologies/FedProClient
- **Released as OSS with permissive Apache 2.0 license**

Conclusion

- **Some advantages of the Federate Protocol**
  - Secure connection to an RTI in the cloud
  - Easy to deploy federates in containers
  - Works with unreliable 4G/5G links
  - Supports additional languages and environments
  - Removes the requirement for RTI-specific libraries in federates

- **Can co-exist with traditional C++ and Java APIs**
  - Optimal for different use cases

Simulation Interoperability Standards Organization

"Simulation Interoperability & Reuse through Standards"

QUESTIONS