Simulation Interoperability Standards Organization

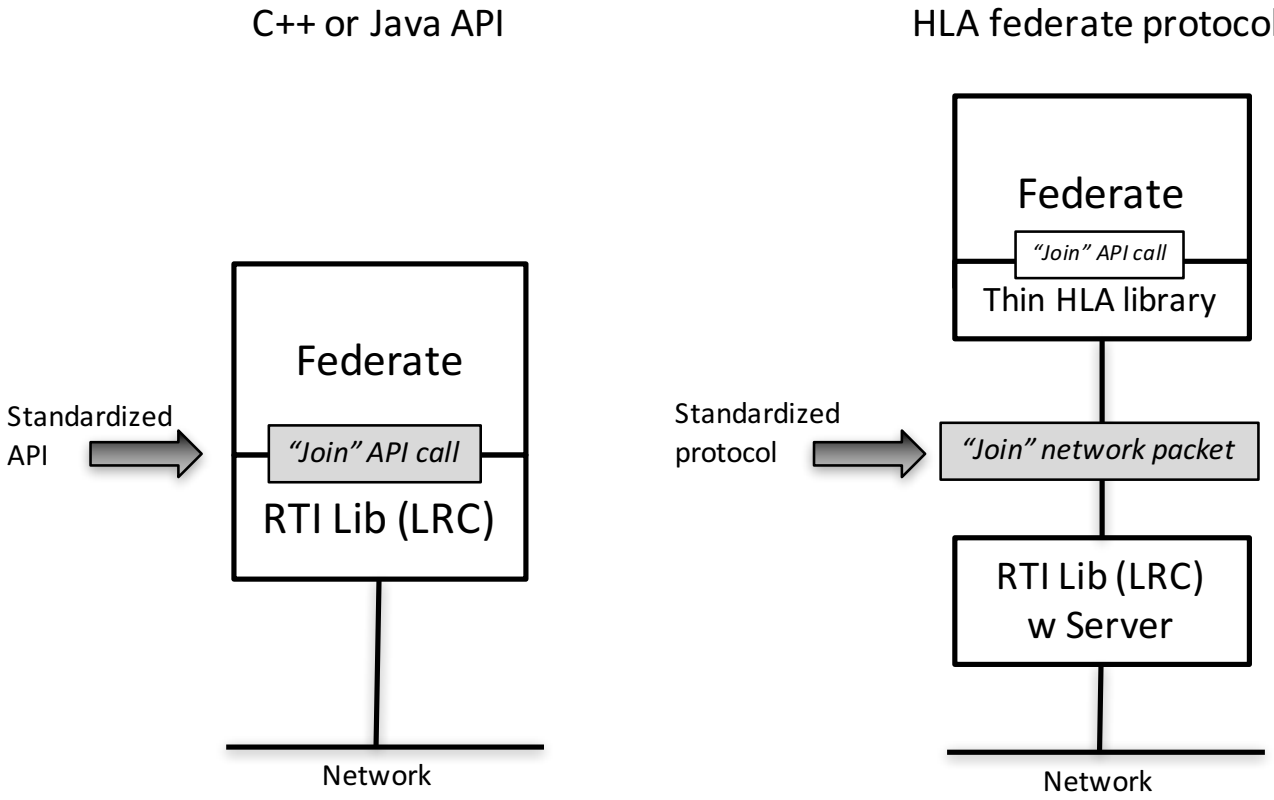
"Simulation Interoperability & Reuse through Standards"

Federate Protocol – From the Drawing Board to the Cloud

2025-SIW-Presentation-002

Fredrik Antelius, Pitch Technologies, Sweden

What is the HLA 4 Federate Protocol?

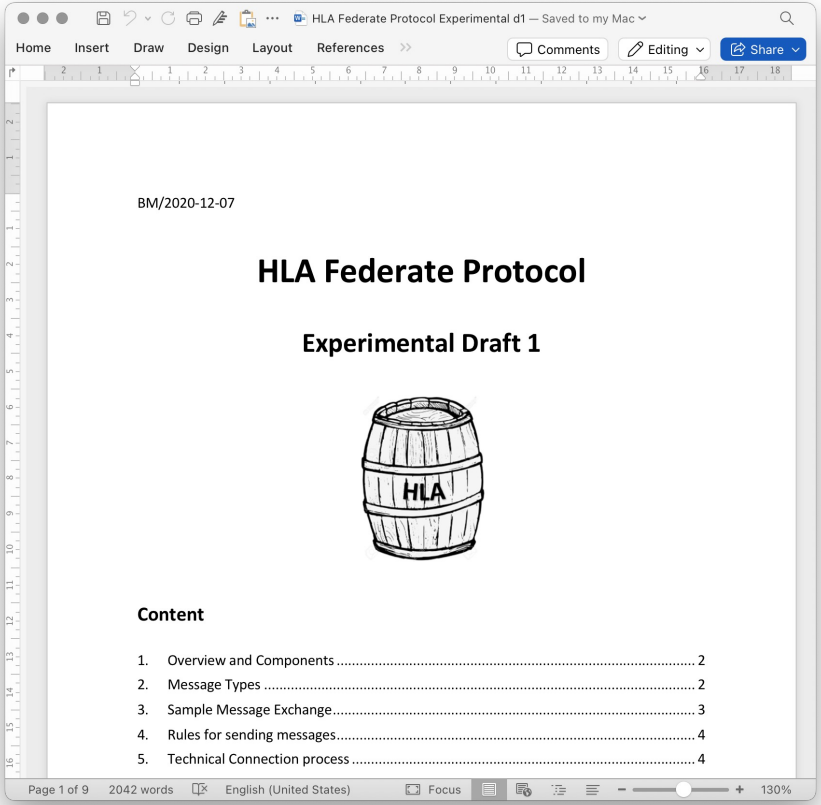


The Beginning – “Thin HLA”

- **“Thin HLA Prototyping Study” – November 2016**
- **Top 3 requirements:**
  1. HLA for other programming language like C#, Python and Swift
  2. No need to replace LRC libraries when updating RTI (“thin LRC”)
  3. Standardized network protocol for HLA
- **Prototype based on FlatBuffers and gRPC**

Specification Timeline

- Comment sent to HLA PDG in May 2017
- Initial draft specification in December 2020
- Focused Tiger Team work throughout 2021
- Incorporated into HLA 4 specification in 2022

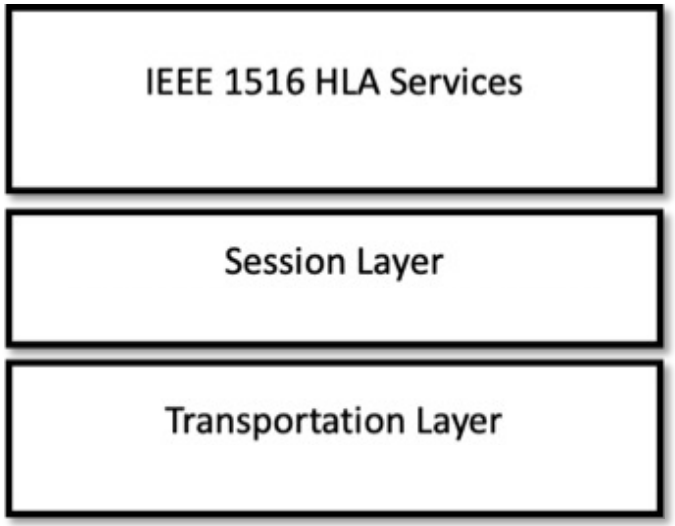


The Drawing board – around 2020

- **Networking that Just Works**
  - Massive, multiplayer online games – Fortnite, …
  - Netflix, web browser, …
- **Suitable for cloud deployment**
  - New networking constrains
- **Resilient connection for live deployments**
  - Simulators at the edge of connectivity
- **Inspectable network protocol**
- **HLA in other programming languages**
  - Embedded devices and constrained environments

Architecture

- **Client–Server model**
  - Federates as thin clients
  - RTI as a central, stateful server
  - No peer-to-peer connectivity
- **HLA Services as network packets**
- **Stack in 3 layers:**
  1. IEEE 1516 HLA Services
  2. Session layer
  3. Transportation layer
- **Based on proven and successful technologies**
  - Don’t reinvent the wheel



HLA Services layer?

- **HLA Services fully defined in Interface specification**
- **Encode service invocation and result into network packets**
  - FlatBuffers, Apache Avro, C-structs, Cap'n'proto?
  - Custom, optimized, format?
  - Protobuf most suitable
    - *Code generation support, multiple implementations and support for many languages*
- **Callbacks are sent immediately**
  - **`evokeCallbacks`** and **`disableCallbacks`** services not available
- **Standardizing API for additional languages was not in scope**

HLA Services in Protobuf

- **Request/response message for service invocation and return value or exception**
  - Both for calls to RTI and callbacks from RTI
- **No optional fields**
  - Different versions of a service have different names
- **Use existing serialization of handles and logical time**
- **FOM files need special handling**
  - Sent as URL, Zip file or raw file content
- **Sets and Maps “shall not contain duplicates”**

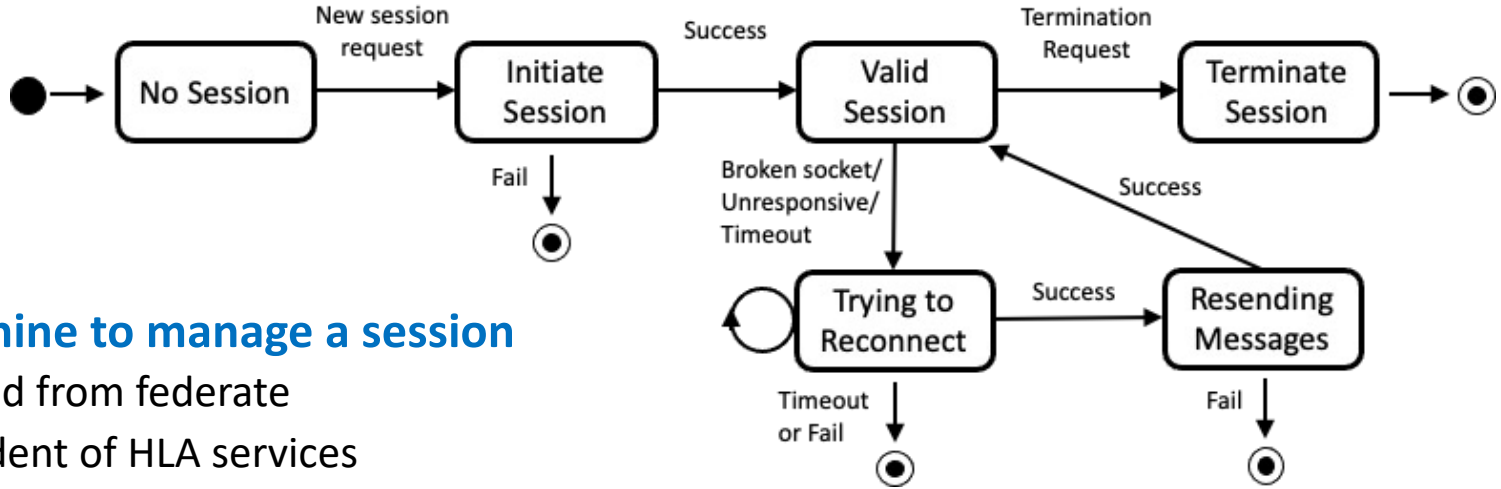
```
message SendInteractionWithTimeRequest {
      InteractionClassHandle interactionClass = 1;
      ParameterHandleValueMap parameterValues = 2;
      bytes userSuppliedTag = 3;
      LogicalTime time = 4;
}

message SendInteractionWithTimeResponse {
    MessageRetractionReturn result = 1;
}
```

Session layer?

- **Resilient, resumable connections with “session cookie“**
  - Network layer connection may be broken
  - Session persist over reconnections
- **Unable to find a suitable protocol**
  - Considered gRPC, SIP and others
- **Use a custom session layer protocol**
  - Designing and specifying a custom protocol was a major effort
- **Different levels of resume support:**
  1. No reconnect or resume
  2. Reconnect and resume last message
  3. Message history with multiple reconnects and full error handling

Custom Session Layer



- **State machine to manage a session**
  - Controlled from federate
  - Independent of HLA services
  - Message history to resend lost messages on resume
- **Control messages**
  - Create, resume, heartbeat and terminate a session
- **HLA service messages**
  - Call and callback request/response

Transport layer

- **Reliable, ordered delivery of Session messages**
  - Responsible for flow control
- **Plain TCP**
- **TLS 1.3 and mutual TLS for encryption and authentication**
  - Encryption
  - RTI authentication
  - Federate and RTI authentication
- **WebSocket and Secure WebSocket**
  - Preferred in enterprise platforms like OpenShift
  - Routing, filtering and load balancing as a HTTP connection
    - *Secured by Ops at the perimeter*

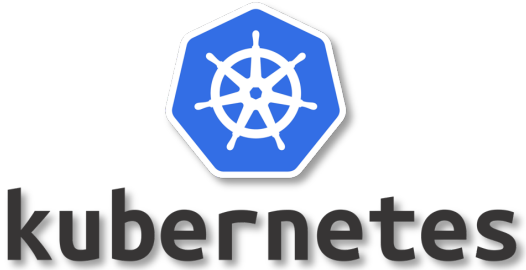
Federate Protocol in 2025

- **Federate Protocol Client library for C++ and Java**
  - Existing federates using HLA Evolved and HLA 4 with C++ and Java APIs
  - Open Source at `GitHub.com/Pitch-Technologies`
- **Federate Protocol Server available in Pitch pRTI**
- **The specification works and can be implemented**
- **Federate Protocol on par with regular APIs**
  - Passes all existing Pitch pRTI tests

- **Built what we had on the Drawing Board**

Just works in cloud & containers

- **No incoming network connections for federates**
  - No UDP and no multicast
- **Single library – no extra dependencies**
- **Treat RTI as any web server or database server**
  - Run Federate Protocol Server as a Kubernetes “service“
    - *Use networking mode (inter-pod, intra-pod, service) depending on deployment*
- **New observability requirements**
  - Tracing, monitoring, diagnostics and telemetry



Existing C++ and Java API over Federate Protocol

- **Synchronous API with blocking calls, like `registerObjectInstance`**
  - All exceptions in RTI calls are blocking
  - Exceptions in Federate callbacks are not blocking
- **Depends on *round trip time* (RTT) between client and server**
  - Breaks some assumptions like instant **`isAttributeOwnedByFedederate`**
  - Exceptional exceptions can be ignored to make calls non-blocking
  - Cache and pre-fetch more state in client
- **New APIs and federate design can mitigate the impact**
  - Non-blocking API with Future and Promise
    - *Non-blocking and blocking APIs can be mixed in a federate*

Additional practical experience

- **Perfect HLA compatibility**
- **Good experience from CWIX -24**
  - Longer startup time
- **Performance impact**
  - Additional encode and decode for each invocation
  - Extra network hop
  - No multicast or peer-to-peer connections
- **Network disruptions happens**
  - Not only at edge of connectivity

Next Steps

- **JavaScript and Python client libraries**
- **Open Source collaboration**
- **Reduce RTT impact**
  - Cache and pre-fetch more state
  - Non-blocking APIs
  - Update federates with “broken” assumptions
- **Improve observability**
- **Performance optimizations**
- **Improve error resilience in Session layer implementation**
  - Based on more operational use

Simulation Interoperability Standards Organization

"Simulation Interoperability & Reuse through Standards"

Q&A / Discussion