# Comparison of High Level Architecture Run-Time Infrastructure Wire Protocols Part Two

Peter Ross
Presented by William Oliver

Defence Science & Technology Organisation
Department of Defence, Australia


2014 Fall Simulation Interoperability Workshop

# Introduction

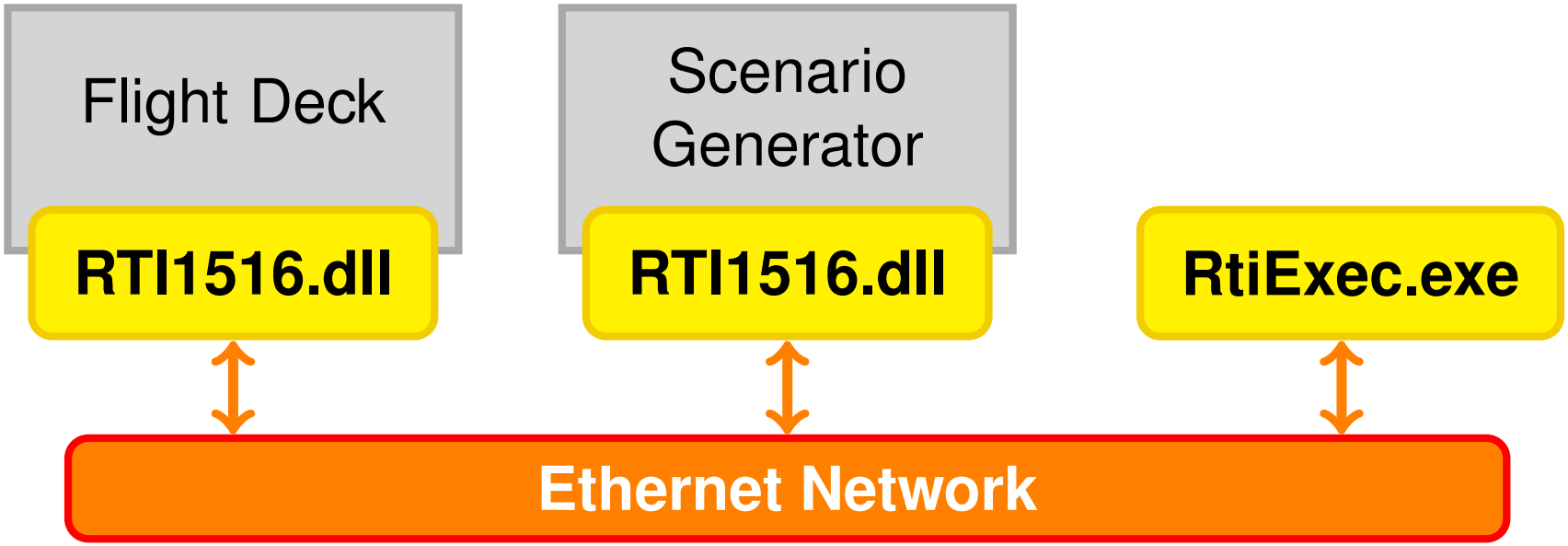
There are many RTI implementations available today, but none can interoperate on-the-wire!

- Why is this so?
- Are wire protocols really different?
- What are the implications?

# Agenda:

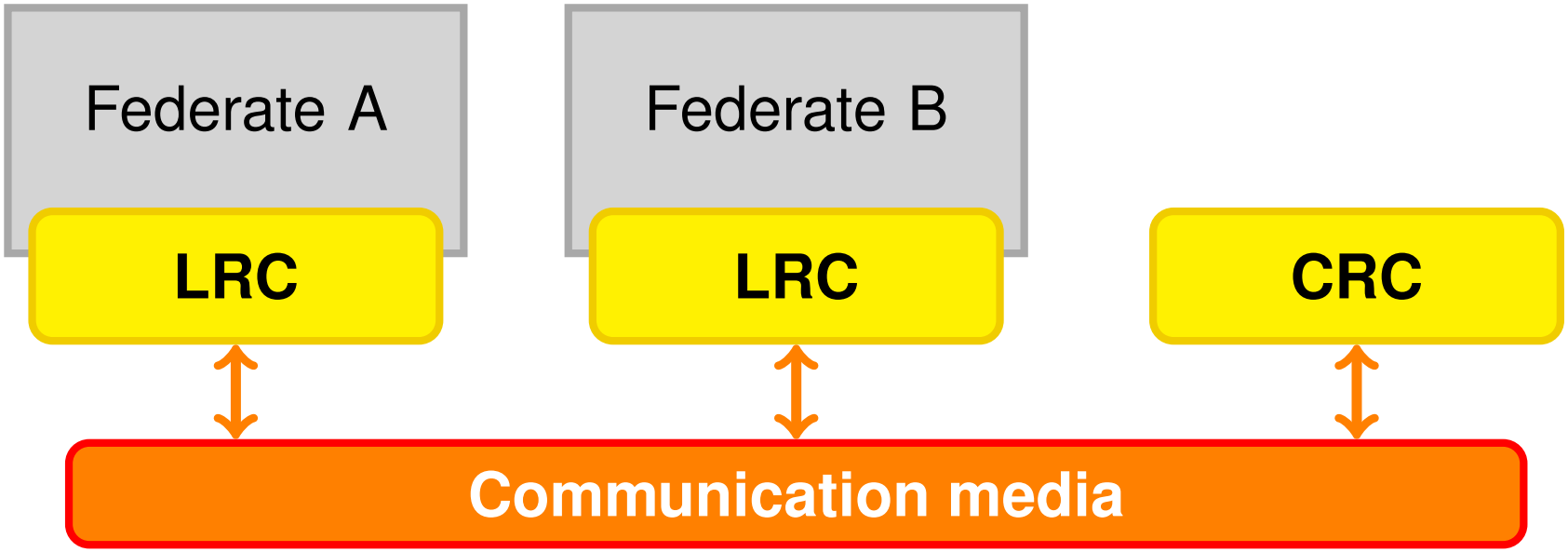
1. Terminology, concepts and motivation
2. Comparison method
3. Results (including findings from *Part One*)
4. Interpretation

# Example Federation



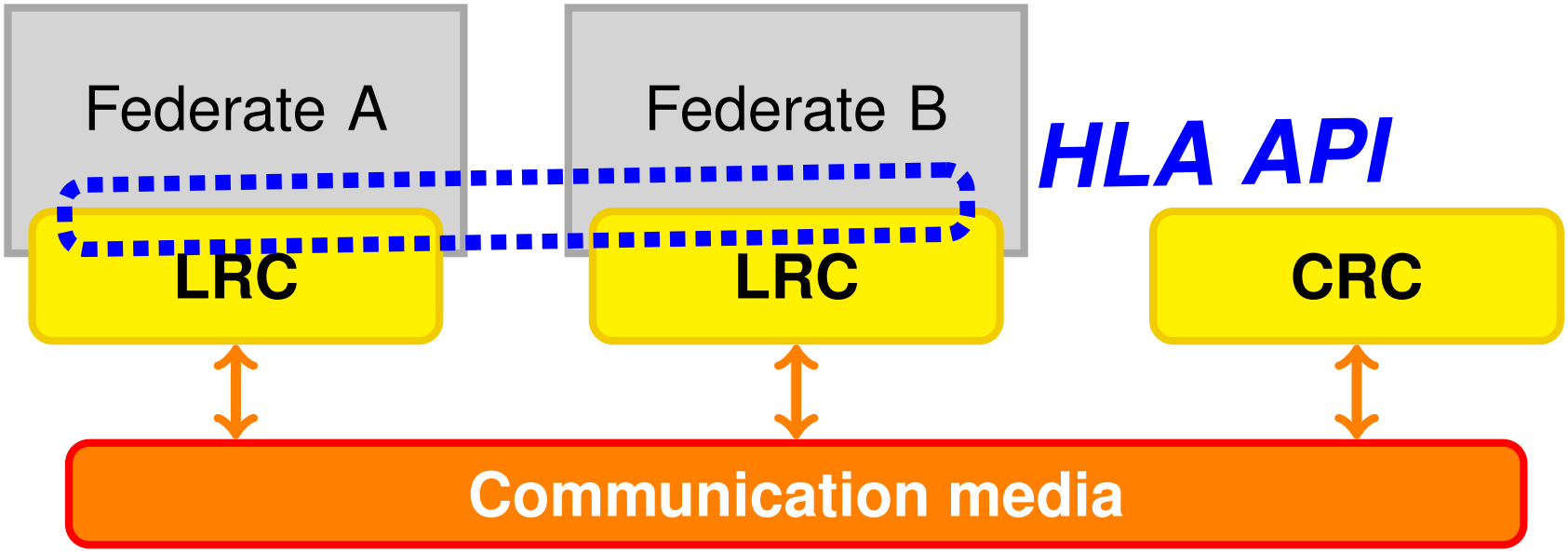

- Flight simulator, two federates
- HLA 1516 middleware installed (yellow components)
- Ethernet connectivity

# Example Federation – Theory



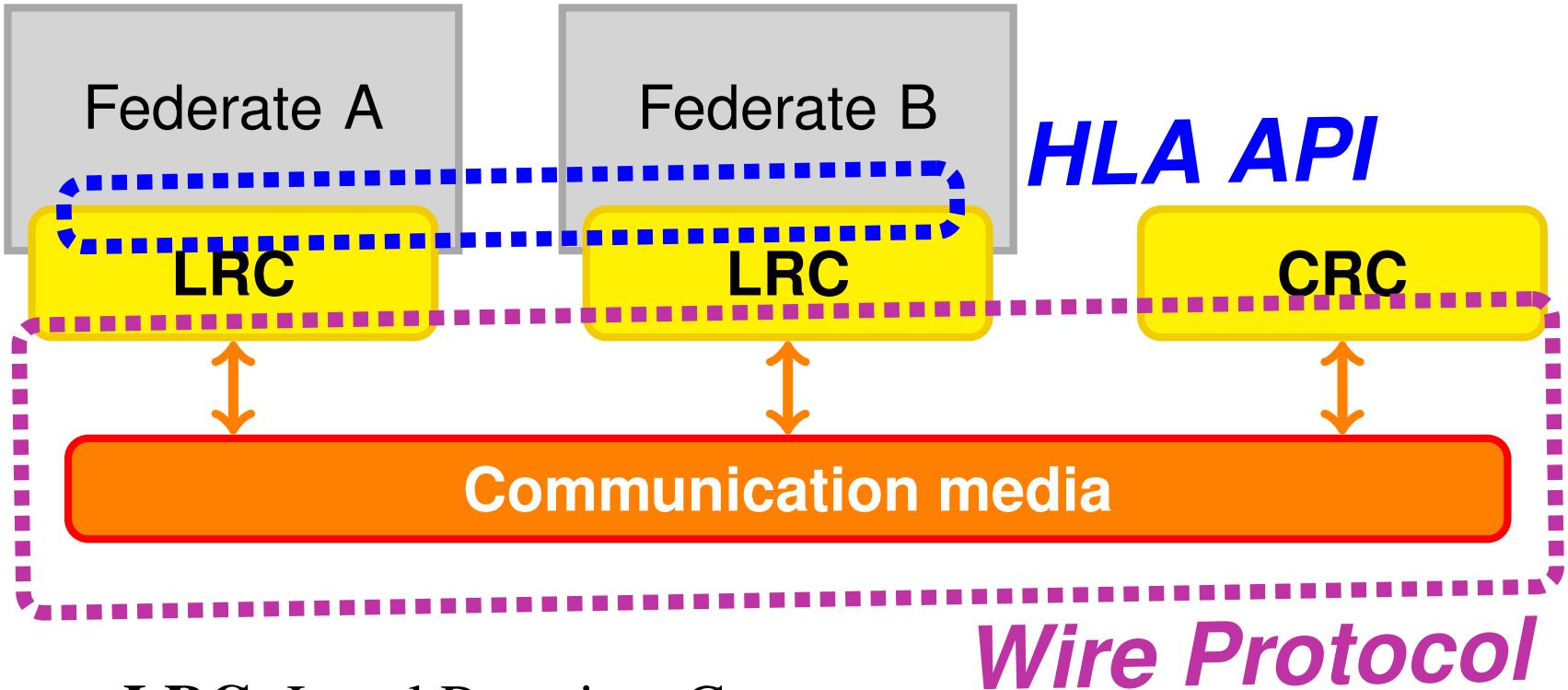

- **LRC**: Local Run-time Component
- **CRC**: Central Run-time Component

# Example Federation – Theory




- **LRC**: Local Run-time Component
- **CRC**: Central Run-time Component
- **API**: Application Programming Interface

# Example Federation – Theory




- **LRC**: Local Run-time Component
- **CRC**: Central Run-time Component
- **API**: Application Programming Interface
- **Wire Protocol**: Establishes how components exchange information

# HLA is an 'API Standard'

> *This document provides a specification for the HLA functional interfaces between federates and the RTI*
>
> 1516.1-2010 §1.3

Federate Interface Specification describes the *requirements*

and programming interfaces (Java, C++, Web Services . . . )

but not how the services are to be achieved

# In practice:

Each RTI implementation uses a proprietary wire protocol

- We cannot mix components from different vendors
- Often we cannot mix *different component versions* from the *same vendor*

# The Problem

All federates must use the same RTI implementation, to guarantee a federation will execute

This is the unwritten rule of HLA!

- Not mentioned in any IEEE standard
- Trial by fire for newcomers

# Why isn't this a formal rule?

HLA is indifferent on wire protocol interoperability

- Implementations are *not required* to *interoperate*
- But they are also *not required* to *not interoperate*

# Current 'Solution'

A priori agreement:

- Decide on a particular implementation
- State implementation name and software version in Federation Agreement

What if you are already using a different RTI implementation?

## Option 1: Change the RTI implementation

Easy to achieve in laboratory → "*just copy some files*"

Less straightforward in real world → technical risk & cost

## Option 2: Use an RTI-to-RTI bridge or gateway

# FedBizOpps.gov

# Technical Risk (2011)

★ FEDBIZOPPS.GOV

5. Rationale Justifying Use of Cited Statutory Authority

This requirement meets the FAR 2.101 definition of a commercial item as these items are available to the general public through Raytheon's GSA website. The Contract Specialist also confirmed the availability with the vendor.

This referenced Runtime Infrastructure middleware is required for continuous efforts on the AN/USQ-T46D BFTT system. This middleware was deployed in the developmental system and is now a required component in the product baseline. Use of a different product would require a re-compile of the federate code in the BFTT system application software and would cause duplication of effort and costs that would not be recovered through competition. The duplication of costs would total approximately $1.25M, which consists of $1,050K for software development, $35K for acceptance testing, $150K for certification testing and $16K for the software development engineer's labor. This duplication would cost the government approximately 14 months for the development effort and an additional 4 months for the acceptance and certification testing. For this reason the requested items are required in order to be 100% compatible with the existing equipment.

*https://www.fbo.gov/index?s=opportunity&mode=form&id=6a347af37763e6175e8e0cf29d1347e9&tab=core&_cview=1*
*http://www.webcitation.org/6S84fVhKK*

# Technical Risk (2012)

★ FEDBIZOPPS.GOV

BRIEF DESCRIPTION OF SUPPLIES OR SERVICES REQUIRED AND THE INTENDED USE

MAK RTI (Run Time Infrastructure) software licenses are required to support Basic Division Officer Course (BDOC) (based on the COVE trainer architecture) training system acquisitions at Naval Base San Diego and Naval Station Norfolk. MAK RTI functionality is an integral component of these training systems. The VShip software is designed to operate with this software, and communication between federates without this software would not be possible. Without this communication, the system would not operate and training would not be possible. The most cost effective acquisition approach is to acquire MAK RTI licenses in a single bulk order and distribute licenses as Government Furnished Software for each independent BDOC trainer initiative.

UNIQUE CHARACTERISTICS THAT LIMIT AVAILABILITY TO ONLY ONE SOURCE, WITH THE REASON NO OTHER SUPPLIES OR SERVICES CAN BE USED

VT MAK is the sole provider of MAK RTI software to the USN fleet (surface and submarine) and associated COVE based training systems. The MAK RTI software is proprietary to VT MAK and is not available from any other source. The MAK RTI is central to the HLA based COVE training system, including all VShip federates. VShip is developed specifically for use with the MAK RTI, and the use of another RTI system will greatly increase development cost, delivery schedule, and result in a significantly increased test program.

EXPLAIN WHY AN ADEQUATE PURCHASE DESCRIPTION OR OTHER INFORMATION SUITABLE TO SOLICIT MORE THEN ONE SOURCE HAS NOT BEEN DEVELOPED.

*https://www.fbo.gov/index?s=opportunity&mode=form&id=9dd53a665a392247e30ba9ee4cf2fe0b&tab=core&_cview=1*
*http://www.webcitation.org/6S8583AJk*

# Progress

RTI Interoperability Study Group (1999)

- On standardising the wire protocol:
  "*while that might be best long-term solution, it might inhibit experimentation and possible development*"

HLA Direct (2003)

- Draft wire protocol by General Dynamics; subset only

OpenRTI Study Group (2004)

- "*At this time there are not the resources to pursue a wire standard, but the concept of a wire standard will remain alive in the community.*"

Every few years somebody throws a grenade on the reflector

Lots of debate, little data

# Research Method

Gather data on existing wire protocol usage

Services of interest

- Support services
- Federation Management
- Object Management
- Time Management

For each implementation:

- Review technical documentation and source code
- Exercise RTI using test federates
- Observe resulting communication with vendor supplied tools and Wireshark
- Identify common and unique methods

# RTI Implementations


BH-RTI


CERTI


HLA Direct


MAK RTI


Open HLA


OpenRTI


Portico

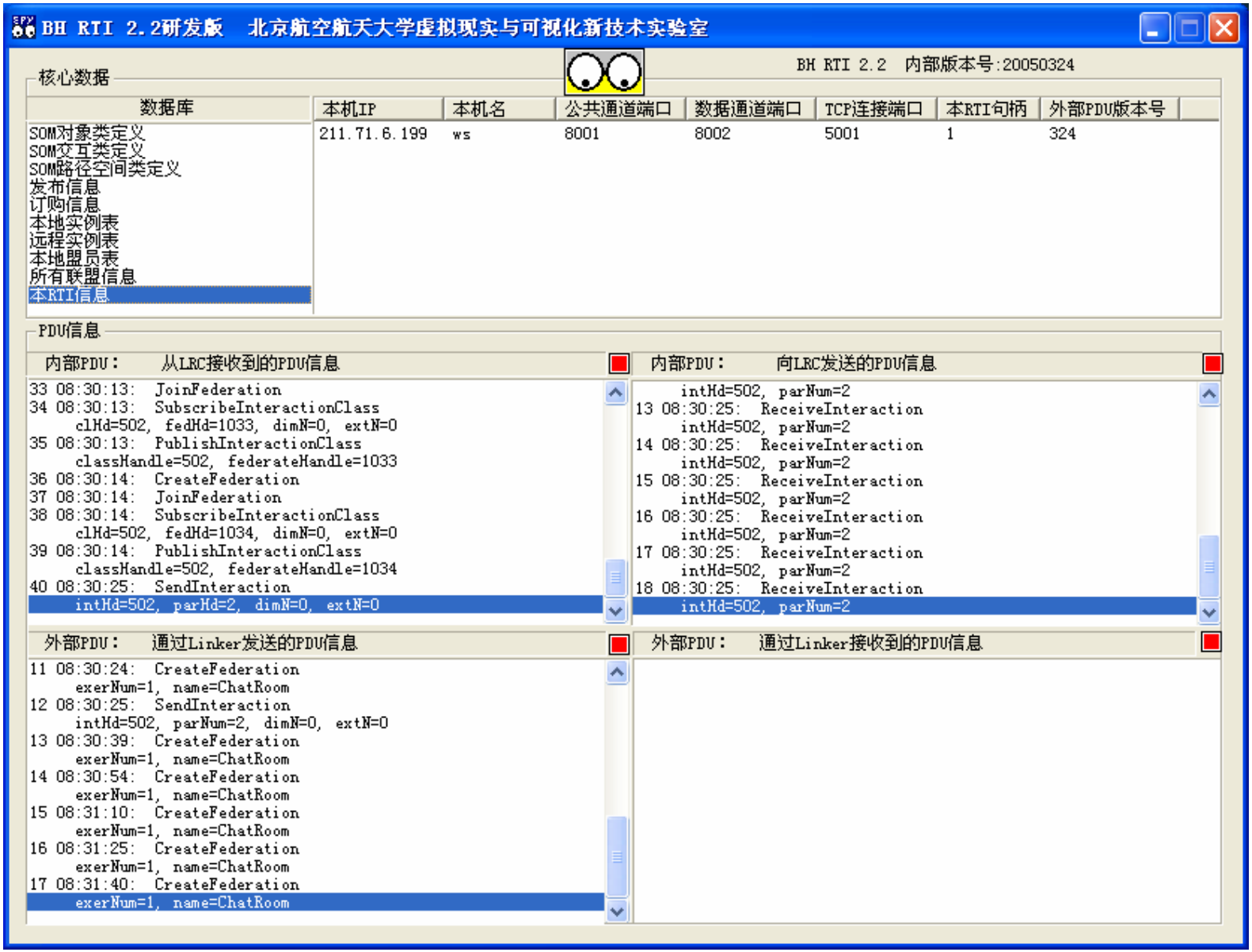

pRTI 1516 (*)


RTI NG Pro (*)


RTI-s

(* = Partial analysis)

# Example Diagnostic Tool – BH RTI



# Example Packet Capture – CERTI

## **Step 1.** Invoke service

```
RTI::AttributeSetFactory::create(1)

updateAttributeValues(
    objectInstanceHandle=0x5,
    attributes={attributeHandle=0x2, value='_ValueA/aa_'},
    tag='_Test-CERTI-001_')
```

endian flag
message length
message type
sizeof(value)
sizeof(tag)

## **Step 2.** Capture resulting packets(s)

```
0000   00 4c 00 00 00 2c 00 00 00 01 00 00 00 01 00 00   .L...,..........
0010   00 00 00 00 00 00 01 10 00 00 00 5f 54 65 73 74   ............_Test
0020   2d 43 45 52 54 49 2d 30 30 31 5f 00 05 00 00 00   -CERTI-001_.....
0030   01 00 00 00 02 00 00 00 01 00 00 00 0b 00 00 00   ................
0040   5f 56 61 6c 75 65 41 2f 61 61 5f 00               _ValueA/aa_.
```

## **Step 3.** Correlate service parameters with packet content

# Results

Part One was presented at SimTecT 2012:

- Component organisation
- Communication systems
- Message formats


Part Two:

- Issuance and receipt rules
- Data structures

# Part One – Mode of Operation

The arrangement of LRCs and CRCs influences the design of the wire protocol

## Decentralised

No CRC


## Centralised

1 CRC

## Hierarchical

$> 1$ CRCs

Two implementations provided a configuration option to specify mode of operation (either decentralised or centralised)

# Part One – Communication Media

What transport protocols do components use?

- Internet protocol **[all implementations]**
- Shared Memory **[2]**
- HTTPS **[1]**
- Some implementations provide ‘software routers’ to extend federation reach across firewalls and proxies. These were not studied.

# Importantly:

Implementations were found to use the same message formats for different media

# Part One – Interconnects

How is the media used?

Decentralised implementations:

- LRC-to-LRC: all used multicast UDP

Centralised implementations:

- LRC-to-CRC: all used TCP
- LRC-to-LRC: three different methods
  1. Multicast UDP
  2. Unicast – also known as ‘fully connected’
  3. Relay – the CRC relays messages between LRCs

Configuration options were often plentiful. We only examined the defaults of each implementation.

# Part One – Message Formats

Proprietary byte-oriented data structures **[all implementations]**

- Header and message body
- Header indicated at least message type and length
- Big-endian **[5]**, little-endian **[1]**, bi-endian **[3]**

Alternative encodings for some services

- Java Object Serialisation **[2]**, CORBA IIOP **[1]**

Other format capabilities:

- Versioning **[5]**
- Fragmentation and reassembly **[3]**, bundling **[5]**
- Sequence numbers **[5]**, checksums **[3]**
- Compression **[2]**
- RTI Initialisation Data (RID) consistency checking **[4]**

# Part Two

We will discuss items in bold today. See paper for full findings.

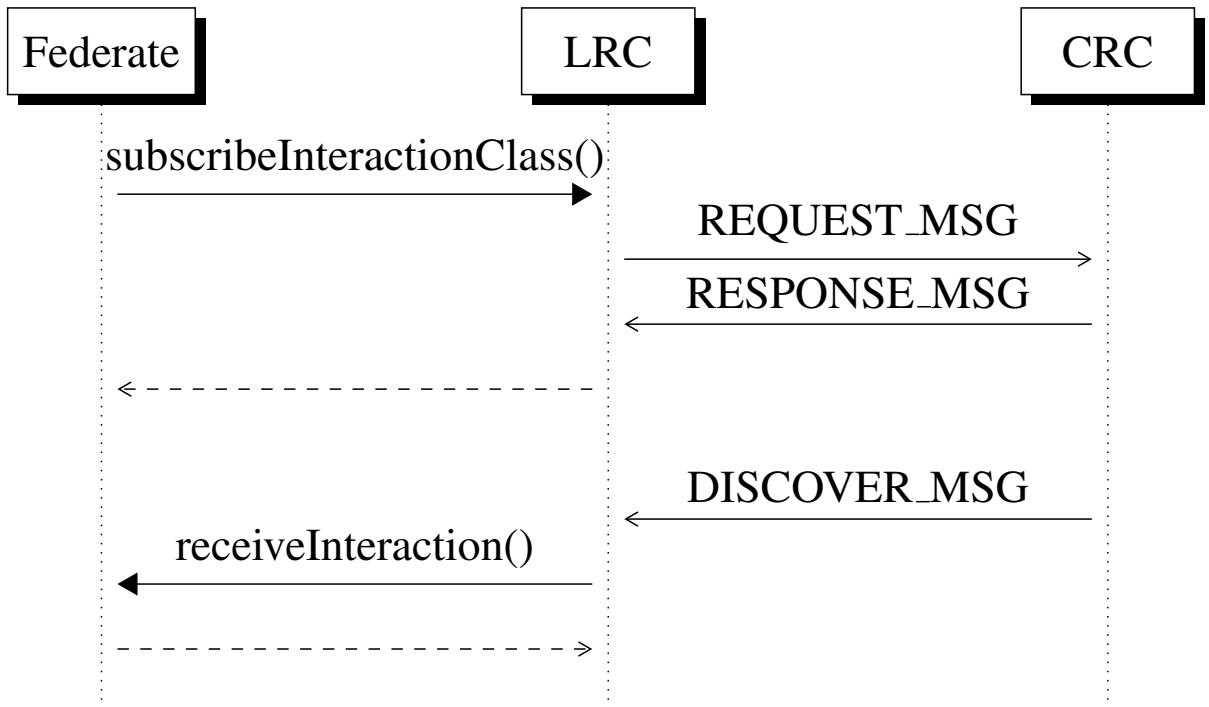
- **Findings for centralised implementations (*)**
- Support services (*)
  - Execution handle, Federate handle
  - Object instance handles, object model handles
- Federation management services
  - Execution name deconfliction
  - Federate name deconfliction
  - File reading and distribution
- Object management services
  - Preventing object name conflicts
  - **Late discovery**
- **Communicating attribute and parameter values (*)**
- Timestamp representation

(* = includes partial analysis)

# Centralised Implementations

All centralised implementations used request and response messages between LRC and CRC.

- Handles ‘on the wire’
- Asynchronous messaging
- Direct LRC↔LRC communication



# Late Object Discovery

When a federate subscribes to an object class, it must discover all existing (and relevant) object instances

How did decentralised implementations achieve this?

- Reactive method **[3 implementations]**
  - When a federate subscribes to an object class, its LRC sends an announcement message to all other LRCs.
  - Other LRCs react, returning list of relevant objects
- Heartbeat method **[1]**
  - The LRC of the owning federate is responsible for sending heartbeat messages
  - Objects are discovered on next heartbeat or update
- Lazy method **[1]**
  - Objects are discovered only when they are updated

# Update Attribute Values Service (1)

> *This service, coupled with the Reflect Attribute Values service, forms the primary data exchange mechanism [of the RTI]*
> 1516.1-2010 §6.10

To achieve this, all implementations used a message containing:

- The object instance being updated
- Attribute handles, value sizes, and value content
- Additional information: communication channel, transport type, user supplied tag

Layout of the attribute data varied:

- Single list of records (next slide) **[4 implementations]**
- Separate lists **[6]**

# Example Update Attribute Values Message

| **Record** | **Field** | **Data type** |
|---|---|---|
| Header | Magic number | $8\times$ uint8 |
| | Message size | uint32 |
| | Message type | enum |
| Body | Federation handle | uint16 |
| | Object instance handle | uint32 |
| | User tag size ($T$) | uint32 |
| | User tag value | $T\times$ uint8 |
| | Transport type | enum |
| | Number of attributes ($N$) | uint32 |
| Attribute #1 | Attribute handle | uint32 |
| | Attribute size ($S_1$) | uint32 |
| | Attribute value | $S_1\times$ uint8 |
| | $\vdots$ | |
| Attribute #$N$ | Attribute handle | uint32 |
| | Attribute size ($S_N$) | uint32 |
| | Attribute value | $S_N\times$ uint8 |

# Update Attribute Values Service (2)

Size of the ‘attribute size’ field

- 32-bit **[8 implementations]**
- 16-bit **[1]**
- 16- or 32-bit depending on M.O. **[1]**

Service invocation did not always result in a single message

- Updates split across reliable and unreliable media
- Did not consider Bundling, DDM or Time Management

# Send Interaction Service

Achieved with near-identical data structure

- Object Class Instance Handle → Interaction Class
- Attribute Record → Parameter Record

# Summary

Mode of operation greatly influences design of protocol

- Centralised implementations: similar, likened to WS API
- Decentralised implementations: more variety

For each service, *at least* one method was shared by multiple implementations.

- Often this was the most obvious design choice
- Novel methods observed
- Absence of sophistication; no Universally Unique Identifiers (UUIDs), no Distributed Hash Tables (DHTs)

*All decentralised* implementations took shortcuts:

- Hash functions
- Random number generators
- Ignoring requirements

# Limitations of Study

Other important services were not studied:

- Save-restore, sync, ownership management and DDM
- HLA-Evolved capabilities

Strength and weaknesses analysis of each method lacking

- Which methods are *best* and *why*?
- Benchmarks

Results apply only to the software versions listed in paper

- RTI implementations and wire protocols are frequently updated

# Conclusions

# Leaky Abstractions

> *All non-trivial abstractions, to some degree, are leaky*
>
> Joel Spolsky

Implementation diversity is a core principal of HLA

- RTI design decisions are left open to the vendor

In absence of specific requirements, vendors are making their own decisions for:

- Handle limits, e.g. max objects per federate
- Maximum attribute and parameter value sizes
- FDD file reading and distribution

These decisions varied across *all* implementations.

How much do they influence *federate development*?

# HLA Rule 4

*During a federation execution, joined federates shall interact with the RTI in accordance with the HLA interface specification*

1516-2010 §5.4

# Rational (informative):

*Federate developers can work independently and develop interfaces to the RTI without regard to RTI implementation*

*and*

*RTI developments can proceed without explicit consideration of federate development.*

1516-2010 §A.1.4

# HLA Rule 4 [Change Request]

*During a federation execution, joined federates shall interact with the RTI in accordance with the HLA interface specification*

1516-2010 §5.4

## Rational (informative):

~~*Federate developers can work independently and develop interfaces to the RTI without regard to RTI implementation,*~~

~~*and*~~

*RTI developments can proceed without explicit consideration of federate development.*

1516-2010 §A.1.4

# Centralised Wire Protocols

Centralised wire protocols were similar because all the hard work is being done by the CRC!

For this kind of RTI, the wire protocol:

1. Translates API calls into compact messages (and back again)
2. Establishes communication channels with LRC peers
3. Sends data directly to LRC peers

Ripe for standardisation

# Decentralised Wire Protocols

The only way to make a high-performance decentralised RTI is to *cheat!*

Decentralised implementations make up half the RTIs studied

None were ‘HLA compliant’

- Hash functions and random numbers were used to approximate uniqueness
- Challenging requirements ignored

The data suggests that it is not feasible to build a compliant RTI without a central server!