Comparison of High Level Architecture Run-Time Infrastructure Wire Protocols – Part One

Peter Ross
Defence Science & Technology Organisation
peter.ross@dsto.defence.gov.au

Abstract. Under High Level Architecture (HLA), distributed simulation services are provided by middleware known as the Run-Time Infrastructure (RTI). Existing RTI implementations, despite being similar in function and performance, do not interoperate with one another as they each use a different proprietary wire protocol. Organisations must therefore agree to use the same RTI implementation when participating in a HLA experiments or exercise. But are proprietary wire protocols really necessary? Is there sufficient similarity amongst RTI implementations to support standardisation of the HLA wire protocol? This paper describes an effort to compare the wire protocols of state-of-the-art commercial, government and open-source RTI implementations. In this paper, Part One, the communication systems and message formats of these RTI implementations are analysed in detail.

1. INTRODUCTION

The wire protocol is an essential part of High Level Architecture (HLA) as it establishes how information is exchanged 'on the wire' between federates. Unlike earlier distributed simulation technologies, HLA standardises the Application Programming Interface (API) between the federate and the middleware, known as Run-Time Infrastructure (RTI). Definition of the wire protocol is left up to the RTI implementation. There are many RTI implementations available today, but because each uses a different proprietary wire protocol, they do not interoperate with one another.

The lack of a wire standard means that all federates must use the same RTI implementation to guarantee that the federation will execute. Consequently, federation agreements specify the RTI implementation name and software version [1]. For large exercises or experiments this often results in some participants having to change their implementation to satisfy the agreement. The changeover process is not without cost or technical risk.

Supporters of the status quo assert that standardising the wire protocol would restrict developer freedom and performance optimisation, whilst opponents assert that interoperability would reduce integration cost and permit better network communication diagnostics [2] [3] [4]. Informed discussion on this issue is made difficult by the proprietary nature of the technology. Little is actually known about HLA wire protocols because developers do not openly publish their specifications.

This is the first of two papers describing an effort to compare the wire protocols of nine RTI implementations (referred to as *implementations* hereon for brevity). The objective of this work is to establish foundations for the development of a standard HLA wire protocol. In Part One, communication systems and message formats are analysed in detail. Part Two shall examine the content of messages, and their issuance and receipt rules.

2. RELATED WORK

The Simulation Interoperability Standards Organization (SISO) has sponsored two standards activities concerning wire protocol interoperability. The *RTI Interoperability Study Group* was formed after the release of HLA V1.3 to explore issues associated with HLA interoperability. It acknowledged that a standard wire protocol was the best long-term solution, but concluded that premature standardisation may inhibit further development of RTI implementations [5]. In the late 1990s, there was a reasonable expectation for further development, as evinced by the many implementations now available. Several years later the *Open Run-Time Infrastructure Protocol Study Group* was formed after one vendor published a draft wire protocol called HLA Direct. The draft supported a subset of HLA services; just enough to permit interoperability between real-time simulators. It was not developed further due to lack of volunteer interest [6]. Both study groups have since disbanded.

The publications associated with some implementations in the late 1990s included the design of their wire protocols, although the detail was often wanting [7] [8] [9] [10]. Since all of these implementations have changed in the decade (or more) following publication, the information is insightful, but not necessarily relevant today. Several authors have compared the distributed computing algorithms that are employed by implementations such as Time Management and Data Distribution Management, but none has examined the corresponding wire protocol messages in detail [11] [12] [13].

3. METHOD

The provision of RTI diagnostic tools by some commercial vendors, and emergence of mature open-source implementations, makes meaningful comparison of HLA wire protocols now possible. A dataset was built characterising the communication systems and message formats of nine implementations. Only the

services defined in the HLA V1.3 interface specification, or the equivalent IEEE 1516 services, were analysed. This enabled a wide variety of implementations to be considered.

For each implementation, the technical documentation and source code were first reviewed to understand its concept of operation and capabilities. Implementations were then evaluated using test federates to exercise the HLA services, and the resulting network communication between federates was captured using the Wireshark[1] network protocol analyser. The capture files were studied to identify message formats and communication channels. Some implementations provided diagnostic tools which display the contents of wire protocol messages in human readable form. These tools aided in the identification of message names and their correlation with specific HLA services.

The implementations and software versions analysed are listed in Table 1. These were chosen on the basis of accessibility to the author, and present a balanced mix of commercial, government and open-source offerings. Further details on each implementation can be found in the open literature. RTI NG, the implementation made freely available by the United States Defense Modeling & Simulation Office (now known as the Modeling & Simulation Coordination Office) was not included as it shares lineage with its commercial successor, RTI NG Pro.

Table 1: RTI implementations analysed; the modus operandi (MO) may be centralised (C), decentralised (D), or hierarchical (H).

Implementation	MO	Version	Release date
BH RTI	H	2.2	2006
CERTI	C	3.4.0	2011
HLA Direct	D	0.1	2003
MAK RTI	C, D	4.1	2012
OHLA	C	0.5	2011
Portico	D	1.0.2	2010
pRTI1516	D	3.2.2	2007
RTI NG Pro	C, D	4.0.4	2006
RTI-s	D	D27D	2012

4. RESULTS

Results are grouped into subsections covering concept of operation, communication system and message format. Algorithms and capabilities found to be shared amongst many implementations are highlighted, as well as unique and novel approaches.

4.1 Modus Operandi (MO)

Run-time Infrastructure is made up of components called Local RTI Components (LRCs) and Central RTI Components (CRCs). The way an implementation arranges its LRCs and CRCs defines its *mode of operation*, or MO, and was found to greatly influence the design of the wire protocol. The LRC is a software library that is linked with each federate process at run-time. It provides an application programming interface to the federate developer, and was named '*librti1516*' or similar. Coordination of the LRCs, if necessary, is performed by the CRC. This took the form of a standalone program for all implementations analysed, and was named '*rtiexec*' or similar.

The MO of each implementation is indicated in Table 1. A *centralised* operating mode is where LRCs are coordinated by a single CRC. A d*ecentralised* operating mode is where LRCs communicate directly with one another, avoiding the need for a CRC. Finally, a *hierarchical* operating mode is a hybrid of the earlier two that supports the deployment of multiple CRCs. At the lower level of the hierarchy each LRC is coordinated by a nominated CRC, and at the higher level each CRC communicates directly with other CRCs. Both MAK RTI and RTI NG Pro provided an option to switch between centralised and decentralised operating modes. Where this switch has an impact on the wire protocol, the operating mode is cited in superscript beside the implementation name (for example, MAK RTI^{C} or RTI NG Pro^{D}).

4.2 Communication Media

All implementations were found to exchange formatted messages between components, and unsurprisingly, all employed Internet Protocol (IP) as the default communication medium. Shared memory (SHM) communication was optionally supported by MAK RTI and Portico, and Hypertext Transfer Protocol Secure (HTTPS) was optionally supported by RTI-s. The *same* formatted messages were exchanged over these optional communication media. This is significant because the perceived difference between IP and SHM communication is often used to justify the lack of interoperability between implementations [2] [5].

Each medium has advantages and disadvantages. IP allows for messages to be sent over different packet-switched infrastructure, such as Ethernet and cellular telephony. It abstracts away the details of the underlying data links, but this prevents any assumption to be made about the quality of service. HTTPS is layered on top of IP and permits messages to more easily flow through firewalls and proxy servers. It is most relevant in home and corporate computing environments. SHM permits federates operating on the same computer to share a common block of memory. Message queues and global variables may exist in this block, offering high throughput and low-latency communication.

[1] Wireshark website – http://www.wireshark.org/

OHLA, Portico and RTI NG Pro were found to depend on third-party network communication libraries, while others used the standard communication services provided by the operating system. The third-party libraries provided functionality relevant to the implementation of HLA services, such as ensuring unique federation execution names. Portico depended on the JGroups[2] reliable multicast library, taking advantage of its channel identification, node addressing and message fragmentation functions. OHLA depended on the JBoss[3] middleware and took advantage of similar functions. RTI NG Pro depended on the Adaptive Communication Environment[4] (ACE) library, but it was unclear which functions were used.

4.3 Communication Channels

Scalability, throughput and latency are key performance indicators of all middleware. Influential here is the way the LRCs and CRCs are interconnected by communication channels. The exchange of administrative messages (such as those associated with time management services) and data messages (such as those associated with the update attribute values service) may be distributed across different channels. Data messages may also be sent on different channels according to *transport type* specified in the Federation Execution Data (FED) file.

The communication channels observed between components for each implementation are summarised in Table 2, with explanation to follow. Note that only the default configuration was analysed. Implementations were found to provide a myriad of interconnect configuration options and additional software-based routing tools. A comparison of the options and tools available was not performed.

Table 2: Communication channels observed between components; channels may be unicast UDP (UU), multicast UDP (MU), and TCP (T); cardinality is indicated in parentheses.

Implementation	LRC-LRC	LRC-CRC	CRC-CRC
BH RTI	-	T	MU(2)
CERTI	-	T	-
HLA Direct	MU	-	-
MAK RTI^{C}	MU	T	-
MAK RTI^{D}	MU	-	-
OHLA	-	T	-
Portico	MU	-	-
pRTI1516	UU, T	T	-
RTI NG Pro^{C}	MU, T	T(n)	-
RTI NG Pro^{D}	MU	-	-
RTI-s	MU(n)	-	-

4.3.1 Decentralised Interconnects

All decentralised implementations employed a similar interconnect that involved multicast UDP communication between LRCs for both administrative and data messages. RTI-s assigned different multicast channels for specific HLA services, while all other implementations used a single multicast channel. There was no consensus on UDP port number or multicast group address.

4.3.2 Centralised Interconnects

Centralised implementations have to concern themselves with two interconnection problems. The first is how to exchange administrative messages between the LRC and CRC, and the second is exchanging data messages between the LRCs.

The first problem was solved identically across all implementations. All employed TCP communication between the LRC and CRC. RTI NG Pro^{C} established multiple TCP communication channels between each LRC and CRC pair, while other implementations used a single channel.

The second problem, the exchange of data messages between LRCs, was solved using the methods outlined below. Some implementation used a combination of the methods.

- *Relay.* The existing LRC-CRC communication channel was *reused* to relay data messages between LRCs. CERTI and OHLA employed this method for reliable and unreliable data messages, while MAK RTI^{C} only used it for reliable data messages.
- *Multicast.* Multicast UDP communication channels were established for exchanging unreliable data messages. MAK RTI^{C} and RTI NG Pro^{C} used this method, and pRTI1516 supported this as an option.
- *Unicast.* Direct communication channels were established between LRCs for the exchange of reliable and unreliable data. This is sometimes referred to as a *fully connected* RTI. Only pRTI1516 and RTI NG Pro^{C} used this method.

Each method offers advantages and disadvantages. The r*elay* method greatly simplifies the communication architecture, but it adds latency, and the CRC can become a throughput bottleneck. *Multicast* makes effective use of bandwidth, but when used on wide area networks, requires special configuration of the network infrastructure, or the use of software-based routing tools. *Unicast* offers reduced latency, and does not require any special configuration, but this comes at the expense of increased bandwidth consumption.

[2] JGroups website – http://www.jgroups.org/
[3] JBoss website – http://www.jboss.org/
[4] ACE website – http://www.cs.wustl.edu/~schmidt/ACE.html

4.3.3 Hierarchical Interconnects

BH RTI employed a hybrid of the decentralised and centralised interconnects. The exchange of administrative messages between LRC and CRC pairs was achieved using a single TCP communication channel. The *relay* method was used for the exchange of data messages. Communication between CRCs was achieved using two separate multicast UDP channels, one each for administrative and data messages.

4.3.4 CRC Discovery

Centralised and hierarchical implementations require a mechanism for the LRCs to discover the network address of the CRC. This requirement is not explicitly stated in the HLA standard, but was observed in all these implementations. MAK RTI^{C} and RTI NG Pro^{C} employed an automated discovery system, where the LRC would send a *discovery request* message on startup to a predefined multicast UDP channel. If a CRC was present on the network, it would respond indicating its network address, and thus enabling a connection to be established between the two components. Other implementations relied on a manual discovery mechanism. Here the network address and port of the CRC were specified in the RTI Initialisation Data (RID) file, or as system environment variables.

4.4 Message Format

For the bulk of communication, all implementations employed a similar message format. Messages were encoded using proprietary byte-oriented data structures, and comprised a common header and message-specific body. For all implementations the header provided fields identifying at least the message type and length. Message types were enumerated, and frequently named after associated HLA services. For example, when a federate invoked the *send interaction* service, the *Send Interaction* message was sent to other federates.

Five implementations used big-endian byte ordering. BH RTI used little-endian byte ordering, while CERTI, HLA Direct and RTI NG Pro supported both ordering types. Bi-endianess was achieved by indicating the message endianess in the first field of the header.

Additional message formats were used by some implementations to complement the proprietary byte-oriented data structures. Portico and pRTI1516 used *Java Object Serialisation* for some, or all, administrative services. This is a facility built into the Java programming language that automates the process of encoding and decoding messages. RTI NG Pro^{C} used the CORBA Internet Inter-ORB Protocol (IIOP) for administrative and reliable data messages.

4.4.1 Other Capabilities

A variety of other capabilities were identified in the message header. There was considerable overlap found between implementations; none were the sole supporter of a specific capability.

Five implementations were found to include a version field in the message header. The purpose of this field was to detect situations when incompatible LRC or CRC software versions were present in the same federation.

Message fragmentation and reassembly are required when sending messages greater than 64 kilobytes over UDP. Three implementations supported fragmentation and reassembly. Those that did not support fragmentation sent large attribute or parameter values over TCP, or otherwise refused to send them.

Five implementations supported message bundling; five implementations supported sequence numbers, for the purpose of identifying dropped messages; and at least three used message checksums to ensure integrity of the message over unreliable communication channels. Message compression was supported by MAK RTI and RTI-s using the Zlib and FastLZ algorithms respectively. Both implementations indicated the use of compression via a flag in the header, and applied compression to the message body only.

Four implementations supported consistency checking of the RID file. This was achieved by LRCs or CRCs sending a checksum of the file in the message header, or the CRC sending an authoritative copy of the file on request. The former method allows configuration mismatches to be detected, while the latter allows LRCs to automatically align themselves to the authoritative file.

MAK RTI and RTI-s supported additional capabilities to perform federation testing. This included federate ping testing, network throughput testing, and remote instrumentation, such as measuring disk and processor utilisation.

5. DISCUSSION

While the nine implementations analysed are inarguably different, the data shows similarity in the design of their wire protocols. This is even more apparent when comparisons are made between implementations sharing the same MO. The significant findings are:

- Internet Protocol was the communication medium of choice for all implementations. Alternate media were not as prevalent, with only three out of the nine implementations analysed supporting them.
- Decentralised implementations all used practically the same interconnect design. Although centralised implementations exhibited less similarity, their use of communication channels was grouped into three methods (relay, multicast and unicast).
- All implementations employed a system of passing formatted messages between components. This system was applied across different communication media, including IP and SHM.
- Messages were encoded using byte-oriented data structures, although some implementations

complemented this with other encodings. Message types were often named after the associated HLA services.

- Other message capabilities, such as versioning and bundling, were supported by many implementations. Message compression was less prevalent.

5.1 Further Work

Thus far the communication systems and message formats of different implementations have been compared, but not the actual content of messages. For example, when a federate invokes the *create object instance* service, what message types are sent and how are they received? Work has commenced answering these questions, by analysing the message content and issuance and receipt rules for popular HLA services. The findings are expected to be published in a follow-on paper.

REFERENCES

1. Tudor, G. & Zalcman, L. (2006), "HLA Interoperability – An Update", SimTecT 2006 Conference Proceedings, pp345-350.
2. Granowetter, L. (2003) "RTI Interoperability Issues – API Standards, Wire Standards and RTI Bridges," 2003 Spring Interoperability Workshop, Paper 03S-SIW-063.
3. Pearce, T.W. & Farid, N.B. (2004) "If RTI's Have a Standard API, Why Don't They Interoperate?" 2004 Fall Simulation Interoperability Workshop, 04F-SIW-100.
4. Mullally, K. et al (2003) "Open Message-Based RTI Implementation – A Better, Faster, Cheaper Alternative to Proprietary, API-Based RTIs?" 2003 Spring Simulation Interoperability Workshop, 03S-SIW-112.
5. Myjak, M.D. et al (1999) "RTI Interoperability Study Group Final Report," 1999 Fall Simulation Interoperability Workshop, 99F-SIW-001.
6. Woodyard, J. & Mullally, K. (2004) "Open Run-Time Infrastructure Protocol Study Group Final Report," 2004 Fall Simulation Interoperability Workshop, 04F-SIW-018.
7. Calvin, J.O. et al (1997) "Design, Implementation, and Performance of the STOW RTI Prototype," 1997 Spring Simulation Interoperability Workshop, 97S-SIW-019.
8. Karlsson, M. et al (1998) "Experiences from Implementing an RTI in Java," 1998 Spring Simulation Interoperability Workshop, 98S-SIW-062.
9. Wood, D.D. & Granowetter, L. (2001) "Rationale and Design of the MAK Real-Time RTI," 2001 Spring Simulation Interoperability Workshop, 01S-SIW-104.
10. Zhou, Z. & Zhao, Q. (2006) "Reducing Time Cost of Distributed Run-Time Infrastructure," Proceedings of the 16th International Conference on Artificial Reality and Telexistence (ICAT 2006), pp969-979.
11. Watrous, B. et al (2006), "HLA Federation Performance: What Really Matters," 2006 Fall Simulation Interoperability Workshop, 06F-SIW-107.
12. Gupta, P. & Guha, R.K. (2007) "A Comparative Study of Data Distribution Management Algorithms," Journal of Defense Modeling and Simulation on Applications, Methodology, Technology, Vol. 4, Issue 2, pp127-146.
13. Chaudron, J-B. et al (2011) "Design and modelling techniques for real-time RTI time management," 2011 Spring Simulation Interoperability Workshop, 11S-SIW-045.