TOWARDS A DISTRIBUTED SIMULATION INTEROPERABILITY (DSI) FRAMEWORK TO ADDRESS INTEROPERABILITY ISSUES FOR SIMULATION PRACTITIONERS

A thesis submitted for the degree of Doctor of Philosophy

By

Athar Nouman

Department of Computing Science,

Brunel University

April 2019

ABSTRACT

Simulations are now used more frequently to conduct experiments on real world or proposed systems, to understand the system behaviour, or for evaluating improvement strategies. Over time, the need has grown for big enterprise businesses to developed sophisticated and complex systems to compete with industry. Also, these businesses are now more connected with each other like a networked enterprise. This has further raised the requirement for developing more and more complex simulations that can interconnect with other businesses. Distributed simulation has been widely used in this context in military applications, but such popularity has not grown in other sectors. The reason behind this is the technical expertise required to establish communication protocols between distributed simulations. There have been efforts by research industry to bridge this gap and the most important work has been development of the High Level Architecture (HLA) standard for providing common communication protocols between distributed simulation models.

The Modelling and Simulation (M&S) industry also provides a lot of literature for developers on modelling standalone simulation. The focus of conceptual modelling in this case has been on model accuracy and efficiency instead of interoperability. This is also discussed in detail in this research. Practitioners have also struggled to find support for underlying technologies until most recently. But with the introduction of standard Runtime Infrastructure (RTI) and simulation development platform support this gap has narrowed.

The HLA standard promised to resolve interoperability issues between distributed simulation models, but only managed to provide standard guidelines up to syntactic level. Therefore, the Simulation Interoperability Standards Organisation (SISO) carried the research forward and identified the interoperability problems faced by practitioners at the semantic level and drew up a list of interoperability issues. However, the published standard SISO-STD-006-2010 only identified the problems but did not provide the semantic solution.

The main contribution of this research has been the Distributed Simulation Interoperability (DSI) Framework that identifies semantic solutions for the interoperability problems listed in the Commercial-off-the-shelf Simulation Package Interoperability Reference Models (SISO-STD-006-2010). This research recommends including these interoperability semantic solutions in HLA Object Modelling Template specifications. By doing so, this will help industry practitioners achieve the interoperability promise made by HLA and make distributed simulation models more re-usable and composable.

ACKNOWLEDGEMENTS

In the name of **Allah**, the Most Gracious and the Most Merciful

Alhamdulillah, all praise to almighty Allah for the strength and the blessing in completing my research and this thesis. Special credit and respect goes to my supervisor, **Dr. Simon J E Taylor**. I like to thank him for his profound guidance, continuous support, patience, encouragement and being a critical friend. I would also like to show my gratitude and respect to my second supervisor, **Dr. David Bell** for his most valued advice and being there when needed.

I would like to specially thank my friend, **Dr. Anastasia Anagnostou**, for her motivation and support for the work we did together and thereafter. She was always available for valuable advice. Also, I would like to extend my thanks to another friend, **Dr. Nauman Riaz Chaudhry**, for his motivation and support.

Further, I would like to thank my family members for their patience and their belief in me. During my research they sacrificed their comfort and leisure to keep me focused. In my family, I first wish to thank my parents, especially my mother who prayed for my success day and night. Secondly, I would like to thank my wife, **Mrs. Saba Athar**, for her patience, understanding, and support. It would have been extremely difficult for me to collect the experiment run data without her support. And finally, I thank my sisters for their prayers and encouragement.

Last, but not least, I would like to thank **Mr. Syed Bakar Ali Zaidi**, the director of my workplace for his support and understanding when I needed it most. And I would like to thank my other colleagues at work for their support, especially **Mrs. Mehroo Humayun**. She not only encouraged me, but she also shared some of my workload to facilitate my focus on research.

I recognise this opportunity as a big milestone in my career and a dream come true. I hope for continued cooperation with all of you in the future.

Thank you all......

DECLARATION

I hereby declare that the research presented in this thesis is my own work except where otherwise stated, and has not been submitted for any other degree.

Athar Nouman

The following papers have been published (or submitted for publication) as a direct result of the research discussed in this thesis:

Nouman, A., Anagnostou, A. and Taylor, S.J.E. (2013) “Developing a Distributed Agent-Based and DES Simulation Using poRTIco and Repast.” *In Proceedings of the 17th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT2013)*, pp. 97-104, Delft, NL, October 30-November 1.

Anagnostou, A., Nouman, A. and Taylor, S.J.E. (2013) “Distributed Hybrid Agent-Based Discrete Event Emergency Medical Services Simulation.” In *Proceedings of the 45th Winter Simulation Conference (WSC14)*, Edited by R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill and M. E. Kuhl, pp. 1625-1636, Washington, DC, December 8-11.

Taylor, S.J.E., Revagar, N., Chambers, J., Yero, M., Anagnostou, A., Nouman, A., Chaudhry, N.R. & Elfrey, P.R. 2014, "Simulation Exploration Experience: A Distributed Hybrid Simulation of a Lunar Mining Operation", Distributed Simulation and Real Time Applications (DS-RT), 2014 IEEE/ACM 18th International Symposium on, pp. 107-112.

Nouman, A., Chaudhry, N.R., Anagnostou, A. & Taylor, S.J.E. 2015, "Investigating an Adaptive Protocol for Distributed Simulation with the SISO-STD-006-2010 Type A.2 Interoperability Reference Model", Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, ACM, New York, NY, USA.

ABBREVIATIONS

ABS	Agent-Based Simulation
ADSO	Australian Defence Simulation Office
ALSP	Aggregate Level Simulation Protocol
AMG	Architecture Management Group
API	Application Programme Interface
BOM	Base Object Model
BPR	Business Process Re-engineering
CBT	Composable Behavioural Technologies
CDDL	Common Development and Distributed License
CH	CSP Handler
COTS	Commercial Off-The-Shelf
CRC	Central RTI Component
CSP	Commercial-off-the-shelf Simulation Package
CSPIF	COTS Simulation Package Interoperability Forum
CPU	Central Processing Unit
DARPA	Defence Advanced Research Project Agency
DDM	Data Distributed Management
DES	Discrete-Event Simulation
DEVS	Discrete-Event System Specification
DIS	Distributed Interactive Simulation
DM	Data Manager
DMSO	Defence Modeling and Simulation Office
DoD	Department of Defence
DSEEP	Distributed Simulation Engineering and Execution Process
DSI	Distributed Simulation Interoperability
DSM	Distributed shared memory
DSS	Distributed Simulation Standards
DVE	Distributed Virtual Environment
EMS	Emergency Medical Services
ETS	Entity Transfer Specification
EXCIMS	Executive Council for Modelling and Simulation
FEL	Future Event List
FDD	FOM Document Data
FIFO	First In-First Out
FMI	Functional Mock-up Interface
FOM	Federation Object Model
GPSS	General Purpose Simulations System
GUI	Graphical User Interface

HLA	High Level Architecture
IEEE	Institute of Electrical and Electronic Engineers
IRM	Interoperability Reference Model
JSIMS	Joint Simulation System
LAN	Local Area Network
LCIM	Level of Conceptual Interoperability Model
LIFO	Last in First Out
LRC	Local RTI Component
LRM	Local Resource Manager
M&S	Modelling and Simulation
M&SCO	Modelling & Simulation Coordination Office
MOM	Management Object Model
NRT	Next Release Time
OMT	Object Model Template
OSA	Open Simulation Architecture
PC	Personal computer
PDF	Probability Density Function
PDG	Product Development Group
RAM	Random Access Memory
RTI	Run-time Infrastructure
SD	Sequence Diagram
SDEM	Simulation Data Exchange Model
SEE	Simulation Exploration Experience
SCT	Semantic Composability Theory
SIMNET	Simulation Network program
SOM	Simulation Object Model
SISO	Simulation Interoperability Standards Organization
SIW	Simulation Interoperability Workshop
TENA	Test and Training Enabling Architecture
TSO	Timestamp Order
V&V	Verification and Validation
VIMS	Visually Interactive Modelling Simulation Model
VIS	Visual Interactive Simulation
WSC	Winter Simulation Conference
XMSF	Extensible M&S Framework

TABLE OF CONTENTS

Chapter 1: Introduction 1

1.1 Overview...1

1.2 Background and Rationale 1

1.3 Research Motivations 3

1.4 Research Aim and Objectives...5

1.5 Research Approach..6

1.6 Thesis Structure...9

1.7 Contribution of the thesis 12

1.8 Summary...12

Chapter 2: Distributed Simulation and Modelling Review...........................13

2.1 Overview..13

2.2 Simulation...13

2.2.1 SIMULATION MODELS.. 17

2.2.2 DISCRETE-EVENT SIMULATION 18

2.2.2.1 DES Approaches...21

2.2.3 SIMULATION SOFTWARE... 26

2.3 Modelling and Simulation...29

2.3.1 SIMULATION STUDIES.. 29

2.3.2 CONCEPTUAL MODELLING.. 36

2.3.3 CONCEPTUAL MODELLING.. 39

2.3.4 ARTEFACTS OF CONCEPTUAL MODELLING 41

2.4 Distributed Simulation..43

2.4.1 INTEROPERABILITY 47

2.4.2 REUSABILITY 49

2.4.2.1 Reusability and Interoperability 52

2.4.3 COMPOSABILITY 52

2.4.3.1 Composability and Reusability 55

2.4.3.2 Composability and Interoperability 55

2.4.4 MULTI-FORMALISM 56

2.5 High Level Architecture 58

2.5.1 INTRODUCTION 58

2.5.2 HLA SPECIFICATION 59

2.5.2.1 HLA Framework and Rules 61

2.5.2.2 HLA Object Model Template (OMT) 62

2.5.2.3 HLA Federate Interface Specification 63

2.5.3 RUNTIME INFRASTRUCTURE (RTI) 64

2.6 Distributed Simulation Interoperability Models 72

2.6.1 INTEROPERABILITY REFERENCE MODELS (IRMs) 75

2.6.2 SUMMARY OF THE RELATED WORK 77

2.7 Summary 81

Chapter 3: Design of Interoperability issues to address IRMs 83

3.1 Overview 83

3.2 Issues in Modelling Interoperability 83

3.2.1 DATA EXCHANGE, COMMUNICATION AND REPRESENTATION 84

3.2.1.1 Data Values..85

3.2.1.2 Data Models..86

3.2.1.3 Schema...86

3.2.1.4 Semantic and Syntactic 86

3.2.2 TIME SYNCHRONISATION 87

3.2.2.1 Conservative Synchronisation 87

3.2.2.2 Optimistic Synchronisation 88

3.2.2.3 Event Synchronisation...88

3.3 Requirements for Interoperability...88

3.4 Research Technique 93

3.5 Simulation Approach...95

3.5.1 DEFINE SIMULATION ENVIRONMENT OBJECTIVES 96

3.5.2 PERFORM CONCEPTUAL ANALYSIS 97

3.5.3 DESIGN SIMULATION ENVIRONMENT... 98

3.5.4 DEVELOP SIMULATION ENVIRONMENT 98

3.5.5 TEST SIMULATION ENVIRONMENT.. 98

3.5.6 EXECUTE SIMULATION... 99

3.5.7 ANALYSE DATA AND EVALUATE RESULTS 100

3.5.8 VERIFICATION AND VALIDATION PROCESS 100

3.5.9 CONCEPTUAL MODELLING AND INTEROPERABILITY...............................101

3.5.9.1 Levels of Conceptual Interoperability 102

3.5.9.2 Conceptual Model for Distributed Simulation 104

3.5.10 DISTRIBUTED SIMULATION PARADIGM 107

3.5.10.1 Discrete Event 108

3.5.10.2 Agent Based..108

3.5.10.3 System Dynamics 109

3.6 IRMs Objectives 109

3.6.1 PROBLEM STATEMENT FOR TYPE A.1: GENERAL ENTITY TRANSFER.......110

3.6.2 PROBLEM STATEMENT FOR TYPE A.2: BOUNDED RECEIVING ELEMENT ..110

3.6.3 PROBLEM STATEMENT FOR TYPE A.3: MULTIPLE INPUT PRIORITIZATION111

3.6.4 PROBLEM STATEMENT FOR TYPE B.1: GENERAL SHARED RESOURCES....112

3.6.5 PROBLEM STATEMENT FOR TYPE C.1: GENERAL SHARED EVENT 113

3.6.6 PROBLEM STATEMENT FOR TYPE D.1: SHARED DATA STRUCTURE.........113

3.7 Summary...114

Chapter 4: DSI Framework..116

4.1 Overview...116

4.2 The Framework 116

4.3 Tools Selection 119

4.4 Data Representation and Distribution..123

4.4.1 DATA REQUIREMENTS..123

4.4.1.1 Contextual Data 124

4.4.1.2 Model Realisation..124

4.4.1.3 Model Validation 124

4.4.2 DATA PROCUREMENT 124

4.4.2.1 Category A...125

4.4.2.2 Category B...125

4.4.2.3 Category C...125

4.4.3 SYSTEM VARIABILITY..125

4.4.4 EMPIRICAL DISTRIBUTION 126

4.4.5 STATISTICAL DISTRIBUTION..126

4.4.5.1 Normal Distribution...126

4.4.5.2 Exponential Distribution...128

4.4.6 DATA SELECTION...129

4.5 IRM Model Conceptualisation...129

4.5.1 CONCEPTUALISATION FOR TYPE A.1: GENERAL ENTITY TRANSFER 129

4.5.2 CONCEPTUALISATION FOR TYPE A.2: BOUNDED RECEIVING ELEMENT....131

4.5.3 CONCEPTUALISATION FOR TYPE A.3: MULTIPLE INPUT PRIORITIZATION.132

4.5.4 CONCEPTUALISATION FOR TYPE B.1: GENERAL SHARED RESOURCES 133

4.5.5 CONCEPTUALISATION FOR TYPE C.1: GENERAL SHARED EVENT............134

4.5.6 CONCEPTUALISATION FOR TYPE D.1: SHARED DATA STRUCTURE..........135

4.6 Summary...136

Chapter 5: IRM Protocols 138

5.1 Overview...138

5.2 Proposed Protocols 138

5.2.1 HLA SELECTION 139

5.2.2 GENERAL ENTITY TRANSFER 141

5.2.2.1 Entity Transfer with Interaction Class Protocol 142

5.2.2.2 Entity Transfer with Attribute Class Protocol..............................146

5.2.2.3 Entity Transfer using null message Protocol 149

5.2.2.4 Model Verification and Validation..150

5.2.3 BOUNDED RECEIVING ELEMENT..151

5.2.3.1 Bounded Receiving Element using Queue Update Protocol 151

5.2.3.2 Bounded Receiving Element using Bounded Buffer Update Protocol 157

5.2.3.3 Bounded Receiving Element using Adaptive Protocol..................162

5.2.4 MULTIPLE INPUT PRIORITIZATION 166

5.2.4.1 Multiple Input Specialised Prioritisation Protocol 166

5.2.4.2 Multiple Input Dynamic Prioritisation Protocol..........................171

5.2.5 GENERAL SHARED RESOURCE...174

5.2.5.1 Continuous Resource Update Protocol.......................................175

5.2.5.2 Next Request Protocol..180

5.2.5.3 Message Queue Protocol 187

5.2.5.4 Next Resource Event Protocol..193

5.2.6 GENERAL SHARED EVENT..197

5.2.7 GENERAL SHARED DATA STRUCTURE..201

5.2.7.1 Central Control Protocol 203

5.2.7.2 Central Update Protocol..210

5.2.7.3 Full Replication Protocol...217

5.3 Simulation Verification and Validation..222

5.3.1 INTEGRATION & INTEROPERABILITY TESTING..223

5.4 Case Study: London Emergency Medical Service (EMS)..................225

5.4.1 DATA COLLECTION 228

5.4.2 REALISATION OF THE MODELS..231

5.4.3 SOFTWARE TOOLS..232

5.4.4 SELECTION OF PROTOCOL..232

5.4.5 SIMULATION TESTING...235

5.4.5.1 Simulation Model Testing...236

5.4.5.2 Interoperability Testing...236

5.4.6 FAULT TOLERANCE...238

5.5 Summary...239

Chapter 6: Evaluation of the Framework 241

6.1 Overview...241

6.2 Testing the simulation environment..241

6.2.1 NETWORK SETTING...242

6.3 Performance Testing...242

6.3.1 GENERAL ENTITY TRANSFER 244

6.3.2 BOUNDED RECEIVING ELEMENT...247

6.3.3 MULTIPLE INPUT PRIORITISATION 253

6.3.4 GENERAL SHARED RESOURCE 255

6.3.5 GENERAL SHARED EVENT 258

6.3.6 SHARED DATA STRUCTURE 260

6.4 EMS Model testing results 262

6.4.1 PERFORMANCE TESTING 262

6.4.2 SCALABILITY TESTING 266

6.5 Revisiting the DSI Framework 267

6.6 Summary 271

Chapter 7: Conclusion 272

7.1 Overview 272

7.2 Thesis Overview and Findings 273

7.3 Research aim and objectives revisited 276

7.4 Framework Value 277

7.5 Research Contribution 281

7.6 Research Limitations 283

7.7 Future Work 284

7.8 Summary 285

References 287

Appendix A I

Appendix B VII

LIST OF TABLES

Table 1: Examples of Simulation Software packages 28

Table 2: Modelling and Simulation area types (Balci, 2016).............................. 57

Table 3: Integration & Interoperability testing 224

Table 4: List of events 229

Table 5: Data collection of check points 237

Table 6: General Entity Transfer Performance 244

Table 7: Bounded Receiving Element PoRTIco Dedicated Network 248

Table 8: Bounded Receiving Element PoRTIco Non-Dedicated Network 250

Table 9: Bounded Receiving Element Pitch Non-Dedicated Network 252

Table 10: Multiple Input Prioritisation Performance...254

Table 11: General Shared Resource Performance...256

Table 12: General Shared Event Performance 259

Table 13: Shared Data Structure Performance..260

Table 14: EMS Distributed Network run 262

Table 15: Single Vs Network Execution run 264

Table 16: Memory Consumption...265

LIST OF FIGURES

Figure 1.1 : Empirical cycle (De Groot, 1961) 8

Figure 1.2: Selected research approach based on empirical cycle......................... 9

Figure 1.3: Roadmap of the thesis 12

Figure 2.1: Elements, relationships, and processes involved in M&S 15

Figure 2.2: Classification of simulation models... 18

Figure 2.3: Events, Processes, Runtime, and Activity 20

Figure 2.4: The executive of four different Simulation Approaches 26

Figure 2.5: Visual Interactive Simulation example for Airport Security 27

Figure 2.6: Steps in Discrete Event Simulation Study (Banks et al., 2009) 31

Figure 2.7: “What if” analysis with simulation (Robinson, 2014) 33

Figure 2.8: Modelling and Simulation Life Cycle (Loper, 2015)........................ 34

Figure 2.9: Distributed Simulation Engineering and Execution Process (DSEEP) (IEEE Std. 1730-2010) 36

Figure 2.10: Simulation studies: Key stages and activities (Brooks and Robinson, 2001) 38

Figure 2.11: Modelling, Validation, and Verification (Tolk, 2010) 38

Figure 2.12: Robinson’s Conceptual Modelling Framework (Robinson, 2014) ... 40

Figure 2.13: The artefacts of conceptual modelling (Robinson, 2011) 42

Figure 2.14: Distributed Simulation... 44

Figure 2.15: Reuse Spectrum (Pidd, 2002).. 50

Figure 2.16: Computable function (Petty and Weisel, 2004) 54

Figure 2.17: Generic vs. Specific component design 55

Figure 2.18: HLA Overview (Nouman et al., 2013) 60

Figure 2.19: RTI middleware communication architecture 66

Figure 2.20: Basic relationship between federate and the RTI............................ 68

Figure 2.21: Strategies of interoperability and reusability 72

Figure 2.22: Interoperability Problem.. 75

Figure 2.23: Reference Interoperability Framework (Taylor, 2006a).................. 79

Figure 3.1: Interoperability Challenge 84

Figure 3.2: Distributed Simulation Engineering and Execution Process (DSEEP) (IEEE Std., 2010) 94

Figure 3.3: Interoperability Phases 102

Figure 3.4: The level of Conceptual Interoperability Model (Wang et al., 2009) 104

Figure 3.5: How simulation model accuracy changes with the complexity of the model (Robinson 2008a). 105

Figure 3.6: Robinson's Conceptual Model Framework revisited 106

Figure 3.7: Approaches in Simulation Modelling (Borshchev and Filippov, 2004) 107

Figure 3.8: Type A.1: General Entity Transfer 110

Figure 3.9: IRM Type A. 2: Bounded Receiving Element.................................111

Figure 3.10: IRM Type A.3: Multiple Input Prioritization 112

Figure 3.11: IRM Type B.1: General Shared Resource 112

Figure 3.12: IRM Type C.1: General Shared Event 113

Figure 3.13: IRM Type D.1: Shared Data 113

Figure 4.1: An abstract view of Distributed Simulation Interoperability (DSI) Framework 117

Figure 4.2: Distributed Simulation Interoperability (DSI) Framework 118

Figure 4.3: LRC Architecture (the poRTIco Project) 121

Figure 4.4: Normal Distribution (Robinson, 2014) 127

Figure 4.5: Normal density function (summary of Pooch and Wall (1993)) 127

Figure 4.6: Exponential Distribution 128

Figure 5.1: Comparison of execution time on a single PC and a network with different numbers of federation models (Nouman et al., 2013) 141

Figure 5.2: General Entity Transfer using HLA 142

Figure 5.3: Sequence diagram (SD) General Entity Transfer via interaction using RTI 144

Figure 5.4 : Activity diagram for entity transfer 145

Figure 5.5: SD General Entity Transfer via attribute using RTI 148

Figure 5.6: Sequence diagram General Entity Transfer using null message 149

Figure 5.7: Activity diagram using null messages 151

Figure 5.8: Bounded Receiving Element using HLA 152

Figure 5.9: SD Queue Update Protocol using RTI 154

Figure 5.10: Activity Diagram for Queue Update Protocol 156

Figure 5.11: SD Bounded Buffer Update Protocol using RTI 159

Figure 5.12: Activity diagram Bounded Buffer Update 161

Figure 5.13: SD Adaptive Protocol using RTI...164

Figure 5.14: Activity Diagram using Adaptive Protocol 164

Figure 5.15: Multiple Input Prioritisation using HLA 167

Figure 5.16: SD Multiple Input Specialised Prioritisation using RTI 169

Figure 5.17: Activity Diagram for Multiple Input Specialised Prioritisation......170

Figure 5.18: SD Multiple Input Dynamic Prioritisation 172

Figure 5.19: Activity Diagram for Multiple Input Dynamic Prioritisation 173

Figure 5.20: General Shared Resource using HLA 175

Figure 5.21: SD of General Shared Resource for Resource Update Protocol 177

Figure 5.22: Activity Diagram of Shared Resource for Resource Update 178

Figure 5.23: SD of General Shared Resource using Next Request.....................183

Figure 5.24 Activity Diagram for General Shared Rsrouces 186

Figure 5.25: General Shared Resource with LRM using HLA 188

Figure 5.26: General Shared Resource with LRM using Message Queue 189

Figure 5.27: SD of General Shared Resource for Next Request 191

Figure 5.28: SD of General Shared Resource using Next Resource Event 195

Figure 5.29: General Shared Event HLA 198

Figure 5.30: SD for a General Shared Event using RTI 200

Figure 5.31: Activity Diagram for Shared Event 201

Figure 5.32: Shared Data Structure using Central Control & Update approach using HLA 204

Figure 5.33: SD for Shared Data Structure using Central Control 207

Figure 5.34: Activity Diagram for Central Control Protocol 210

Figure 5.35: Sequence Diagram of Central Update Shared Data Structure.........214

Figure 5.36: Activity Diagram for Central Update Protocol..............................216

Figure 5.37: Shared Data Structure using Full Replication approach.................217

Figure 5.38: SD of Full Replication Shared Data Structure...............................220

Figure 5.39: Activity Diagram Full Replication 222

Figure 5.40: Ambulance Service timeline (Fitzsimmons, 1973)........................226

Figure 5.41: A&E department flowchart diagram (Anagnostou, 2014). 227

Figure 5.42: London Emergency Medical Service Model................................230

Figure 6.1: Execution time for General Entity Transfer proposed protocols 245

Figure 6.2: Comparison for General Entity Transfer proposed protocols...........246

Figure 6.3: Comparison for Bounded Receiving Element using PoRTIco on a Dedicated Network..249

Figure 6.4: Comparison for Bounded Receiving Element using PoRTIco on a non-dedicated Network..251

Figure 6.5: Comparison for Bounded Receiving Element using Pitch on a non-dedicated Network..253

Figure 6.6: Execution time for Multiple Input Prioritisation proposed protocols 254

Figure 6.7: Execution time for General Shared Resource protocols...................256

Figure 6.8: Comparison of different shared resource protocols 257

Figure 6.9 : Execution time for General Shared Event protocols 259

Figure 6.10: Execution time for shared data structure protocols 261

Figure 6.11: EMS Distributed Network Run 263

Figure 6.12: Single Vs Network Execution run 264

Figure 6.13: Scalability run results 267

Chapter 1: Introduction

1.1 Overview

The research presented in this thesis proposes a framework to address interoperability issues in distributed simulation. The emphasis of this research is to support the Commercial-off-the-shelf Simulation Package (CSP) vendors and other Simulation practitioners to take advantage of the proposed framework based on the Interoperability Reference Models (IRMs) to address the interoperability issues that are not covered by the Institute of Electrical and Electronic Engineers (IEEE) 1516 standard. This research will also help practitioners to better address interoperability issues in reusability and scalability for large scale models.

This introductory chapter provides a summary background study to help the reader understand the rationale behind the research. This section will briefly introduce the reader to the building blocks for this research followed by the research motivation for developing the framework to address interoperability issues for distributed simulation, and the aim and objectives of the research. A research approach is discussed to fulfil the aim and objectives presented in the earlier section. Finally, the chapter closes with a chapter summary followed by the thesis structure, including a diagrammatic view which gives the reader a pictorial illustration of the thesis structure.

1.2 Background and Rationale

The increase in computational power due to technology enhancements and low cost hardware has resulted in introducing new prospects to the Modelling and Simulation (M&S) industry. According to Pierre L'Ecuyer, large and more complex models are now possible to simulate because of the possibility of parallel and distributed simulation via new hardware technologies such as multicore processors, cloud computing, and graphics processing unit (Taylor et al., 2013).

M&S has been accepted as the most frequently used methodology for solving real world problems for more than two decades (Taylor et al., 2006). In many industries, M&S techniques are recognised as a critical technology to enable many core systems engineering functions (IEEE 1730.1, 2013). The increase in complexity and scalability of the models and the advent of new networking technology with supportive protocols has led to extensive use of distributed simulation. Distributed simulation is defined as a process "that enables a simulation program to execute on a computing system containing multiple processors, such as personal computers (PCs), interconnected by a communication network" (Fujimoto, 2000). The distributed simulation approach presents several advantages, such as integration of simulation models, cost effectiveness, reusability, encapsulation, etc. Distributed Simulation also has some challenges, mostly related to issues concerning modeling and interoperability (IEEE 1730.1, 2013; SISO-STD-006-2010) especially in large scale models. Interoperability is an important concept for distributed simulation, termed as "the ability of a simulation model to provide services to and accept services from other simulation models, or simulation model related components with the important goal to make an effective operating environment." (Vasilecas et al., 2010).

A generic HLA framework was developed to facilitate interoperability and reusability by the Defence Modeling and Simulation Office (DMSO) for the Department of Defence (DoD), defined by IEEE 1516. The development of HLA standard was also limited because different approaches were used by different CSP vendors and other simulation practitioners to interact their models (Santos et al., 2013). This use of different approaches and implementations led Simulation Interoperability Standards Organization's (SISO) Commercial off-the-shelf (COTS) Simulation Package Interoperability Product Development Group (CSPI PDG) to introduce a set of patterns to address complex interoperability problems, defined in the Standard for CSP Interoperability Reference Models (SISO-STD-006-2010). The proposed framework presented in this thesis presents an effective implementation for interoperability issues in distributed simulation as defined in the IRMs (Taylor et al., 2012a).

The rationale behind this research is based on the understanding that distributed simulation is still presented with challenges for the Operational Research/Management Sciences (Taylor et al., 2013). The distributed simulation technique was introduced to address the need for more complex and large-scale models for industry, which raised some challenges. Usually these models were designed using COTS Simulation Packages. The biggest challenge faced by practitioners of distributed simulation techniques was the interoperability issue between models. Although IEEE 1516 Standard promised to bring solutions to all interoperability issues, but it failed to address all interoperability issues at semantic level. The standard for CSP IRMs is the latest advancement to address these interoperability issues. The concept of this thesis is that as the IRMs have helped practitioners to identify interoperability issues and bridge the gap between distributed simulation and IEEE 1516 standard, similarly, the framework presented in this thesis will bridge the gap for practitioners on how to best address these IRMs.

The hypothesis presented in this thesis will define an effective framework to implement the issues identified by Standard for COTS Simulation Package Interoperability Reference Models and how this protocol will be effectively adopted by HLA standard to support Commercial Simulation Package interoperability.

1.3 Research Motivations

HLA was designed and developed by the DMSO for the US DoD, for integrating different types of their existing simulations (Kuhl, et al., 1999). The involvement of the SISO has largely supported development of distributed simulation in the M&S industry.

A majority of simulation practitioners and COTS vendors still find distributed simulation difficult and are hesitant to adopt it because they lack the proper tools to overcome its disadvantages. This thesis is intended to take this research a step forward for simulation practitioners and COTS vendors by answering the question raised by Boer et al., (2006), "How can we make distributed simulation and existing distributed solutions, like HLA, more attractive to the industrial community?"

This research is motivated by advancements made in distributed simulation standards (DSS). It was discovered by Boer et al., (2008) that distributed simulation is rarely used in industry and mostly used in military defence, mainly because of the semantic interoperability problem in the HLA specification. HLA was designed by the DoD for two main reasons: interoperability and reusability. The reusability concept was introduced to integrate existing models with other existing models or new models. Interoperability and reusability were not much of a problem for DoD simulation models, but other simulation practitioners from industry found difficulties in implementation because of existing interoperability issues with HLA standard. Later, SISO presented a set of patterns of these interoperability problems. Therefore, it is logical to produce a framework to address these patterns and simplify the semantic interoperability problems raised by industry.

A further motivation for this research is the contribution towards composability, where composability is defined as "a state ensuring the consistent representation of truth in all participating systems" (Tolk, 2013). Syntactic (engineering), Semantic (modelling) and Pragmatic are three different levels of composability defined by Gutiérrez and Leone (2013). Syntactic Composability is related to simulation component communication and Semantic Composability "is a question of whether the models that make up the composed simulation system can be meaningfully composed" (Weisel et al., 2003). Pragmatic Composability simulation components are aware of the intent of the use of data. Therefore, it is justifiable to say that this research also enhances syntactic and pragmatic composability by introducing fault tolerance strategies between the simulation components.

1.4 Research Aim and Objectives

The aim of this thesis is to investigate how to address interoperability issues in the distributed simulation environment to benefit simulation industry, and to provide recommendations (if any) for improvement in the HLA standard. To achieve this aim, the following objectives were underlined.

Objective 1: *Present the hypothesis and identify the interoperability issues as defined in the IRMs.*

Initial research indicated a gap in resolving interoperability issues in a distributed simulation environment with no semantic guidelines available for simulation practitioners. These interoperability issues are highlighted in IRMs. A framework will be proposed using the detail, understanding, and knowledge gained from the literature reviews. The problem will be broken down by addressing each individual IRM separately.

Objective 2: *Develop an understanding of underlining standards, techniques, and industry approaches to identify if interoperability issues are hindering the use of distributed standards.*

For the development of a framework, thorough understanding of underlining technologies needs to be investigated; this would include the relevant published standards, tools, packages, simulation development environments and different implementations. From this study a final hypothesis and aim could be listed.

Objective 3: *Propose a framework to address interoperability issues.*

To assess effectiveness and limitations of the framework, it will be evaluated practically and theoretically. The framework will be empirically implemented using generic case studies. Refinements to the framework will be proposed based on the results of this evaluation. Implementation of the framework will reveal any limitations that could not be identified by theoretical evaluation.

Objective 4: *Experimentally test the Distributed Simulation Interoperability (DSI) Framework.*

Generic case studies for each IRM will be developed to experiment and collect test results for evaluation. These experimentations will allow reflection upon the limitations and consolidate the individual IRM solutions, which will lead to the final framework.

Objective 5: *Evaluate the performance of the DSI framework and test the hypothesis.*

The framework will be evaluated based on the performance of related proposed solutions in the industry and the suitability of the framework for implementing the IRMs. Based on this evaluation, it will be decided if the hypothesis is accepted or rejected.

1.5 Research Approach

There are two possible directions adopted by researchers to build and test their theories (Creswell, 2003). The "inductive approach" begins with the wider world view and heads towards abstract generalisation. It funnels from the detailed specification to a simpler generalisation. This approach is also sometimes known as "theory generating". The "deductive approach" starts with the abstract and moves towards concrete empirical evidence. This approach is also known as "theory testing". Using the deductive approach (empirical study), a hypothesis is generated from the theory and research and later the hypothesis is tested by applying research methods (Bryman and Bell, 2007). Collecting empirical evidence in computer science usually follows four clear steps – generating a hypothesis, identifying method, compiling results, and conclusion (Johnson, 2003).

The field of research involves two distinctively known research methodologies known as Quantitative and Qualitative. Qualitative research focuses on describing situations using theories and research tools like observation, interviews and survey, with the focus on generating theories (Saunders et al., 2007; Bryman and Bell, 2007). This research methodology emphasises an inductive approach. Qualitative research uncovers trends in opinions and thought and digs deep into a problem. Quantitative research is a more logical data-oriented methodology. This methodology quantifies the numerical data or data collected via laboratory experiment, paper survey, mobile survey, face to face interviews, observation, system studies, or numerical methods and uncovers patterns in the research. Quantitative research emphasises the deductive approach.

The choice of research methodology depends on the research involved and the hypothesis. It is noted that researchers often combine qualitative and quantitative methods to better answer research questions.

Empirical research is yet another popular method among scientific researchers, which relies on observation and experimentation. In other words, it is research using empirical evidence. Empirical evidence (objective evidence) can be analysed by either qualitative method or quantitative method. According to Hughes (2016), "Quantitative research is empirical research where the data are in the form of numbers. Qualitative research is empirical research where the data are not in the form of numbers". Adriaan De Groot has presented a clear understanding of the empirical cycle, as shown in Figure 1.1. The empirical cycle has five stages as listed below (De Groot, 1961).

1. Observation: Generating the hypothesis
2. Induction: Formulating the hypothesis
3. Deduction: determining consequences of the hypothesis
4. Testing: Testing the hypothesis with empirically derived evidence
5. Evaluation: Evaluating the outcome.

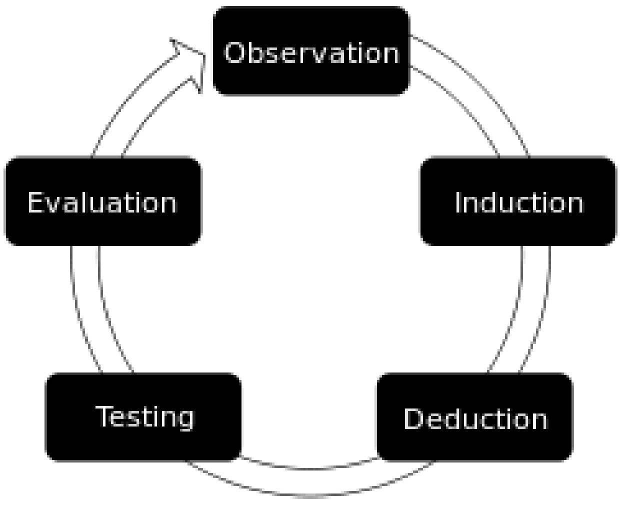


Figure 1.1 : Empirical cycle (De Groot, 1961)

It is observed that scientists use both quantitative and qualitative methodologies collectively. To achieve the research objective, i.e., to address the interoperability issues defined by the IRMs, an inductive approach has been applied. First of all, primary scientific knowledge needs to be gathered regarding M&S, distributed simulation, standards, etc. The literature will be obtained from different published sources. On the basis of the summarised information, a deductive approach is applied to define the hypothesis of the research. The following are the questions identified to test the hypothesis and to address the research question.

1. Summarise the literature and develop the hypothesis.

2. Build generic case studies using simulation techniques for appropriate IRMs.

3. Validate the individual simulation model by comparing the findings with those in the literature.

4. Evaluate each case study results and revisit the framework.

To refine the framework and its development, the case study method is used. A case study is a method of observing in a systematic way (Weick, 1984). According to (Yin, 2009, p. 13) “A case study is an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between the phenomenon and context are not clearly evident”. In brief, empirical methodology is used in this thesis and the steps are defined in Figure 1.2.

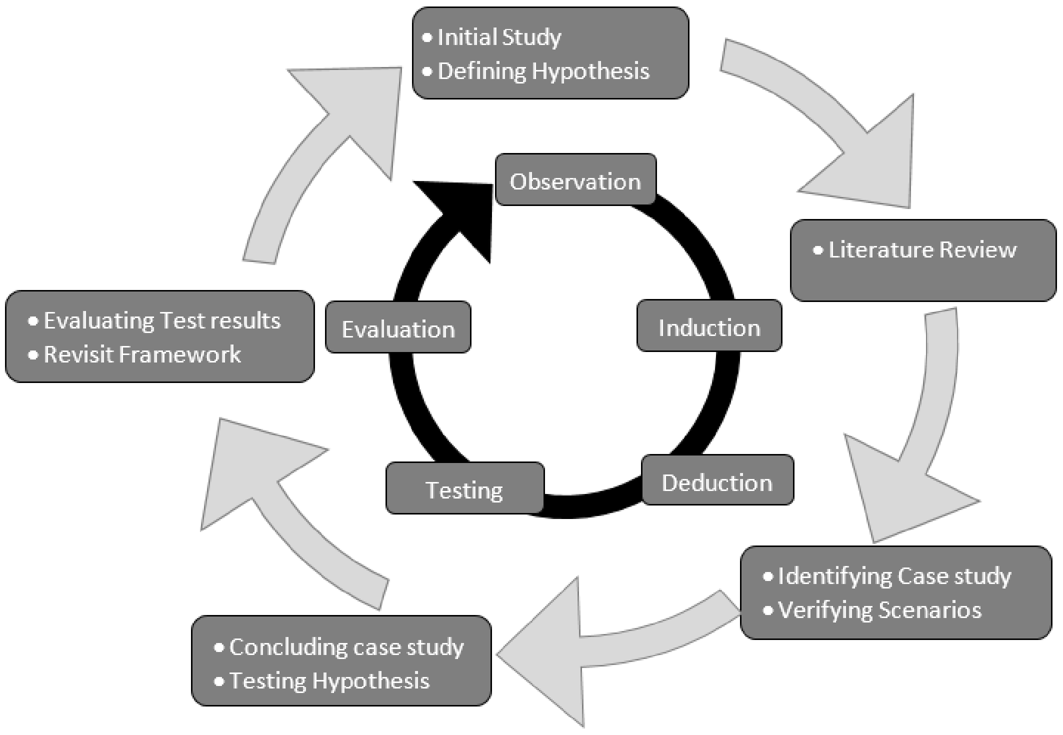


Figure 1.2: Selected research approach based on empirical cycle

1.6 Thesis Structure

The roadmap of this thesis, which is divided into seven chapters, is illustrated in Figure 1.3.

Chapter 1 is an introduction to the work carried out in this thesis, underlining the research motivation and objectives. The first half of the chapter presents a brief background to the proposed research, followed by the hypothesis expressing the need for a framework to address the interoperability issues. Furthermore, this chapter discusses the selection of research methodology and thesis structure.

Chapter 2 discusses the background theory and the relevant literature. This chapter starts with some background discussion about computer simulation and distributed simulations, followed by certain industry standards. This chapter then goes into detailed discussion about interoperability, reusability, composability and multi-formalism in distributed simulation and the benefits of using a standard approach.

Later in this chapter, HLA is explained with its advantages and its limitations. The last section of this chapter focuses on the Interoperability Standard SISO-STD-006-2010, which is published by the Simulation Interoperability Standards Organisation (SISO). This chapter will provide enough knowledge to the reader to understand the need for a framework.

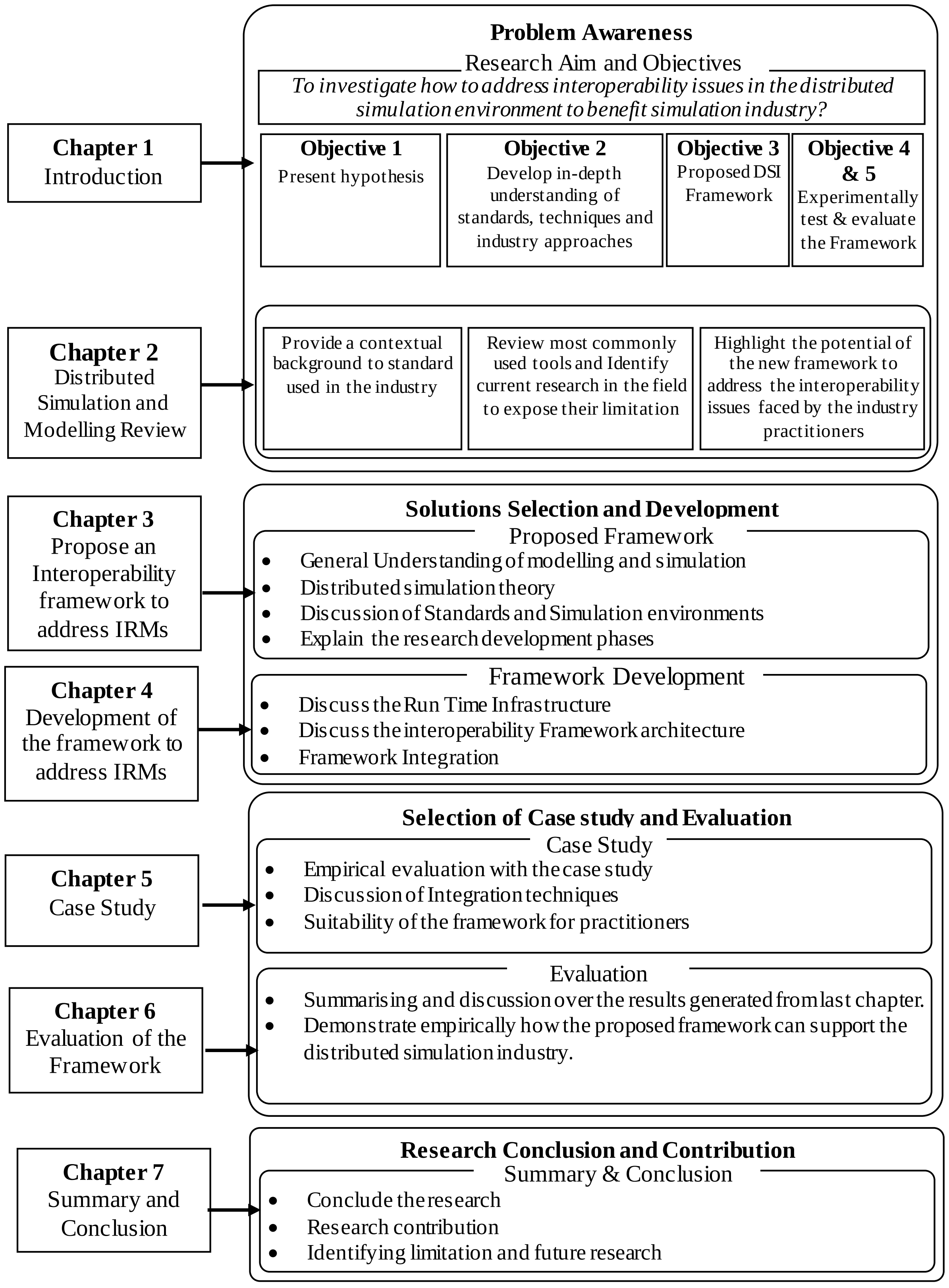
Chapter 3 is focused on presenting the interoperability framework requirements and some limitations. The framework refers to each IRM pattern. This chapter will also explain how this framework could be integrated with simulation models and how it can promote reusability. Further this chapter discussed a use of distributed simulation modelling methodology. Finally, the chapter concludes with an argument for needing a framework based on the interoperability issues highlighted by IRMs.

Chapter 4 discusses development of a framework based on the discussions in Chapter 3. This chapter also discusses the suitability of this framework for the simulation practitioner from industry and presents conceptual models.

Chapter 5 presents the framework design by deploying a generic case study based on the conceptual models prepared in the previous chapter. This chapter is important for identifying the limitations of the framework, which cannot not be done without implementation. Further, this chapter discuss a case study for London Emergency Medical Services (EMS).

Chapter 6 discusses the results collected from the experiment runs on the case studies identified in the previous chapter. This chapter will evaluate the interoperability framework and discuss the usability of the framework. Finally, the framework is revisited based on the evaluation.

Chapter 7 presents the summary and the conclusion of the thesis. It identifies the research contribution and discusses how the aim and objectives have been achieved. Based on the evaluation and revisit to the framework in the previous chapter, the hypothesis is accepted. This chapter also discusses future areas of research in the field of distributed simulation.



Chapter 1
Introduction
Chapter 2
Distributed Simulation and Modelling Review
Chapter 3
Propose an Interoperability framework to address IRMs
Chapter 4
Development of the framework to address IRMs
Chapter 5
Case Study
Chapter 6
Evaluation of the Framework
Chapter 7
Summary and Conclusion
Problem Awareness
Research Aim and Objectives
To investigate how to address interoperability issues in the distributed simulation environment to benefit simulation industry?
Objective 1
Present hypothesis
Objective 2
Develop in-depth understanding of standards, techniques and industry approaches
Objective 3
Proposed DSI Framework
Objective 4 & 5
Experimentally test & evaluate the Framework
Provide a contextual background to standard used in the industry
Review most commonly used tools and Identify current research in the field to expose their limitation
Highlight the potential of the new framework to address the interoperability issues faced by the industry practitioners
Solutions Selection and Development
Proposed Framework
General Understanding of modelling and simulation
Distributed simulation theory
Discussion of Standards and Simulation environments
Explain the research development phases
Framework Development
Discuss the Run Time Infrastructure
Discuss the interoperability Framework architecture
Framework Integration
Selection of Case study and Evaluation
Case Study
Empirical evaluation with the case study
Discussion of Integration techniques
Suitability of the framework for practitioners
Evaluation
Summarising and discussion over the results generated from last chapter.
Demonstrate empirically how the proposed framework can support the distributed simulation industry.
Research Conclusion and Contribution
Summary & Conclusion
Conclude the research
Research contribution
Identifying limitation and future research

Figure 1.3: Roadmap of the thesis

1.7 Contribution of the thesis

After establishing the interoperability issues faced by the practitioners while distributing the simulation models, different protocols were proposed to address these interoperability challenges. After conducting experiment runs on these proposed protocols they were evaluated and a propose solution was concluded to address the interoperability issues presented by IRMs. This research also uncovered some other contribution which are listed below:

1. The major contribution of this research is the DSI Framework, to enable simulation modellers and vendors to satisfactorily address the interoperability requirements of their distributed simulation environments.
2. The second contribution of this research is to facilitate capturing of interoperability requirements at a semantic level.
3. The third contribution of this research is the identification of the recommendation for new callback mechanisms in HLA Standard.
4. The final contribution made by this research concerned making models more integratable and reusable.

1.8 Summary

This chapter began with an overview of the chapter contents followed by the background and rationale in Section 1.2. The next section defined the research aim and objectives. A detailed discussion was presented on which research methodology is to be selected for this research and why. Finally, this chapter presents an overview, or a roadmap, of the thesis to the reader.

The next chapter presents the underlying standards and detailed background relevant to the proposed interoperability framework.

Chapter 2: Distributed Simulation and Modelling Review

2.1 Overview

This chapter provides a contextual background to basic concepts, methodologies, tools, and standards involved in simulation modelling as well as the specific interoperability challenges in developing models for distributed simulation. The topics examined in this chapter begin with a review of the purposes of simulation and the different approaches and software involved. This is followed by further explication of M&S, distributed simulation, high-level architecture and the problems of interoperability faced by distributed simulation modellers. The chapter concludes with a review of various attempts to respond to outstanding distributed simulation problems as defined by the formal IRMs.

2.2 Simulation

According to Shannon (1975), simulation is

> "the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system."

Simulation can alternatively be described as a means of representing a real world system using an artificial environment. It is commonly used as a method for conducting experiments, whether to test existing systems or to develop new methodologies. Simulation is also widely used in training programmes to familiarise personnel with, for example, business processes. It can be effective in providing initial training in many spheres to help avoid human errors and to provide role-playing practice in collective endeavours. A significant function of simulation-based training is also to provide a safe environment in which to practice risky or dangerous operations, such as in the medical profession, the military domain, or any situation that presents health and safety risks. As noted by Banks (2010), simulation models mimic real-world systems, but they require the application of actual historical data to represent the characteristics of the systems they seek to reproduce.

M&S, is a term which is also in common use and in fact defines the complete process more accurately. Modelling is first undertaken to create a representation of the system under focus. Simulation follows as the process of "running" the model to determine how the system performs and reacts. Together, M&S performs the re-creation and analysis of real-world physical systems. M&S is a powerful method of examining aspects of the real world using a scaled-down artificial environment (Ingalls, 2013). It enables analysis of real systems that can help reveal the impact upon such systems of change.

Employing M&S to study physical systems may occur for a variety of reasons. It may just be too expensive to create a real physical system or impractical to study an actual system due to inconvenience or physical endurance barriers. In the academic sciences today, M&S is used as a tool in very many fields of study, including defence, medicine and healthcare, computer communications, transport systems, the ecological environment, biosciences, behavioural studies, economics, and many more. It is also used widely in commercial domains, such as in the simulation of business practices involved in manufacturing and production, and in determining the provision of complex consumer services such as those in the retail financial sector (Taylor et al., 2012).

Topçu et al. (2016) depicted M&S as involving a series of elements, relations, and processes with results derived from determining the interaction between those elements. This is shown in Figure 2.1.

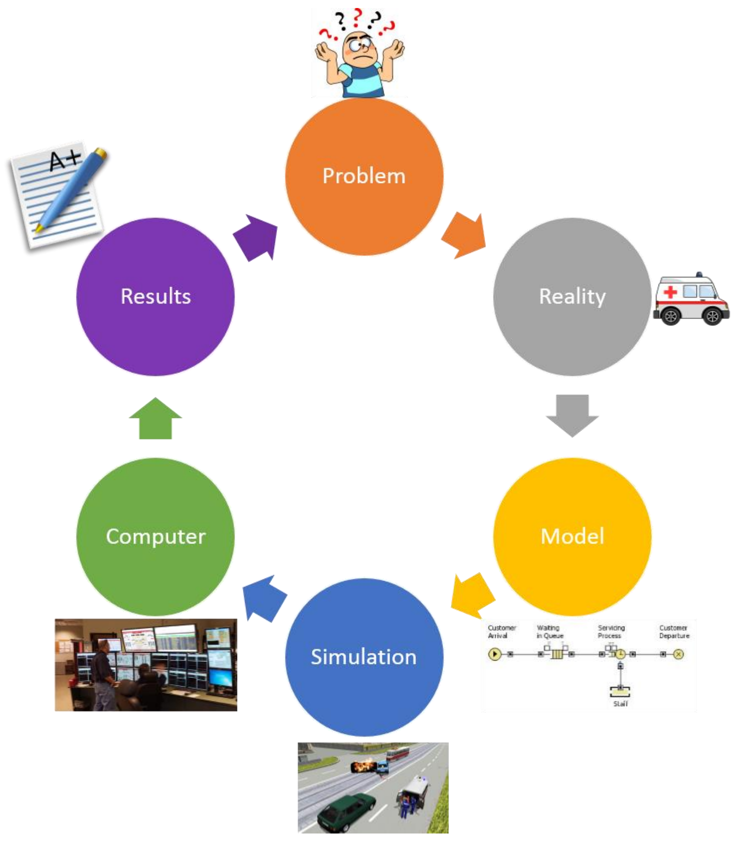


Figure 2.1: Elements, relationships, and processes involved in M&S

M&S may be used to create changes to real systems that already exist or to develop new systems so that resources can be used most effectively and performance is optimal (Maria, 1997). M&S may also be employed when a real system is not available, perhaps because it is inaccessible, access is dangerous or inconvenient, or a real world system does not exist.

Computer simulation employs the computing power to run experiments on models that represent systems of concern (Pidd, 2004). Using computer simulation for experimentation is more cost-effective than using a real system, particularly when trial experiments may be discarded. Computer simulations can also be conducted and deliver results much more quickly than with real world systems. Analysing the behaviour of systems is easier with a simulation, and experiments can be readily replicated on a computer platform. The simulation also provides a safe environment for the study of dangerous scenarios (Brooks and Robinson, 2001). Use of a computer model for testing increases knowledge about the system under review and helps establish appropriate procedures for operating the system (Shannon, 1998).

Modelling is used to re-create a complex real-world system that can then be run virtually as a simulation. Because the computer simulation is virtual, it is essential to establish a detailed system view at modelling stage (Fishwick, 1995).

Computer simulation is used for a variety of purposes in many domains. In industry, simulation can help improve productivity by determining the best deployment of resources (Zimmers and Brinker, 1978). In finance and insurance industries, simulation can be used to analyse portfolio assets and liabilities (Herzog and Lord, 2003). Military forces can use simulation to improve training, rehearse critical missions, and test and evaluate strategy (Page et al., 2004). Healthcare authorities can simulate the outcomes of unpredictable epidemic events and model complex delivery systems such as blood supplies (Lowery, 1998). Simulation can be used to study human performance by integrating human and system performance models (Laughery, 1998). Business process re-engineering (BPR) can also benefit from simulation modelling to determine the effects on performance of process redesign (Bhaskar et al., 1994).

Simulation that is based upon a single execution unit, such as a Central Processing Unit (CPU) core, is known as sequential simulation. Sequential simulation involves “pipeline” processing, or the processing of events in non-decreasing timestamp order (TSO). As each event is executed, state variables and the global clock are accordingly updated. Sequential simulation offers simplicity, but it is constrained in managing complex and very large simulation models. A single processor CPU may be easily overwhelmed by a large volume of events and Random Access Memory (RAM) capacity may be inadequate for retaining complete state information (Lopez et al., 2006). Additionally, some real-world events are not subject to modelling in sequential time.

“Big simulation” is an expression describing the scale of data input for execution, and output for analysis, of large connected simulation models running on a highly distributed computer platform. Although employing standard internet tools and protocols, such situations require ontologically defined models and model interfaces and large relational and non-relational databases to create models and produce output. Dynamic model partitioning and robust domain decomposition is also required (Taylor et al., 2013b).

2.2.1 Simulation Models

Models can be either a mathematical or a physical representation of a system and "a simulation model is a type of mathematical model of a system" (Banks et al., 2009). Simulation models are further classified into being static or dynamic, discrete or continuous, and deterministic or stochastic. A static model is a system represented at a particular point in time, while a dynamic model represents a system that alters over time. Static simulation models are also known as Monte Carlo simulations. Similarly, simulation models having no random variable (as input) are called deterministic. A system is known to be deterministic if its behaviour is entirely predictable. Deterministic simulations work as a deductive form of science, e.g., the operating cycles of an automated machine, which keep repeating the same steps over and over again with the same output.

A simulation model having one or more random variables (as input) is called a stochastic simulation. Stochastic simulation works as an inductive form of science; an example would be the production of electricity from a hydroelectric dam. The amount of electricity produced depends on the scale of water pressure and how it hits the propellers of the turbine. There can be many variables that might affect the water pressure, such as water level, hard vs. soft water, water temperature, weather, etc. Discrete simulation models or systems are those in which "the state of the variable(s) changes only at a discrete set of points in time" (Banks et al., 2009), whereas continuous simulation models are those in which the state of the variable(s) continuously changes over time (Banks et al., 2009). However, representation of a continuous system does not necessarily require a continuous simulation model, nor does representation of a discrete system necessarily require a discrete simulation model. In fact, several modelling approaches could be used together. The decision on modelling approach depends on objective and system characteristics. Since this research is conducted using distributed Discrete-Event Simulation (DES), further discussion and research presented in this thesis is based on discrete, dynamic, and stochastic approaches.

Figure 2.2 illustrates a hierarchical chart to explain the relationship between these different simulation models.

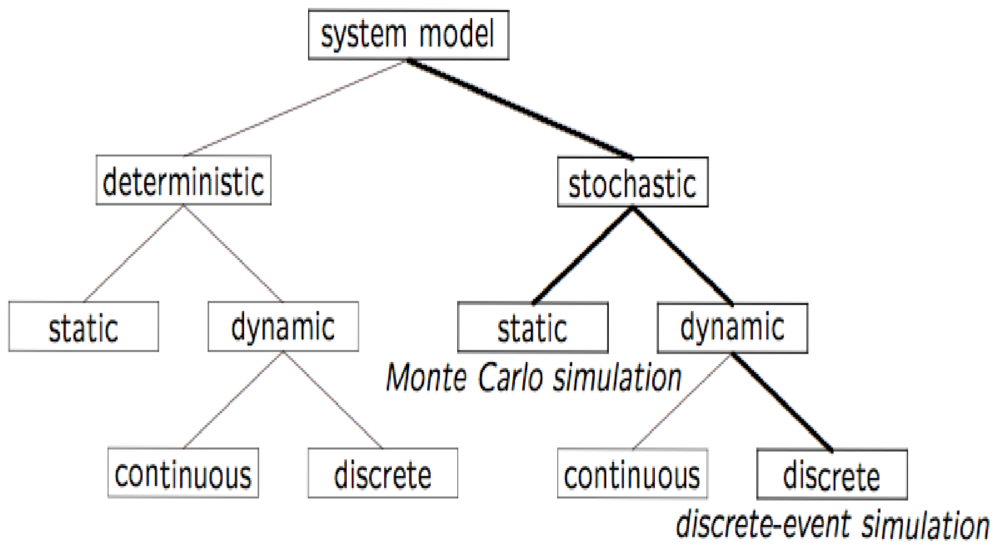


Figure 2.2: Classification of simulation models

2.2.2 Discrete-Event Simulation

As described above, discrete simulations are those systems in which "the state of the variable(s) changes only at a discrete set of points in time" (Banks et al., 2009). Pidd (2006) defines DES as "a discrete simulation that employs a next-event technique to control the behaviour of the model". An example might be a queuing system outside a kiosk. Here, customers stand in a queue before each customer independently orders food. The customer joins the queue from the back and will leave once served. Depending on the order, the length of serving time will be discrete. Therefore, the simulation will proceed at certain events such as when a customer joins the queue, places the order, is served the order, and then leaves the queue.

Before going into further details of DES it is important to understand the major concepts of some terminologies used in DES practice.

- **Model:** An abstract representation of a system (e.g., kiosk ordering as mentioned in the above example)
- **System:** A process containing a number of entities that interact over time to achieve a goal (e.g., a call centre, logistics, and as in the kiosk example, a process of serving customers).
- **Entity:** An object that requires specific representation in the simulation model (e.g., a customer, tyre, medicine, etc.). There is a further classification of types of entities, i.e., permanent and temporary. Permanent entities remain in the system until the end of the simulation, while temporary entities leave the system once they have no further interest.
- **Attributes:** These are the properties of an entity (e.g., customer height or weight). They represent extra information about an entity.
- **Classes:** Entities are individuals, but similar entities of the same type can be represented by classes (e.g., customers, tyres).
- **Resources:** These are representative of individual system elements and are part of the model. They can represent either a system or a human resource (e.g., a machine operator for the latter). These resources may affect the simulation process therefore the simulation keeps count of them at all times. (E.g., in the above kiosk example, persons preparing the food or serving the customers are considered as resources while the customers are entities.)
- **Queue(s):** A logical or physical location where associated entities are collected and the ordering is maintained by some logical function (e.g., First In–First Out (FIFO) or by priority). These queues can be placed at the input of any process, as described in the kiosk example above.
- **Event:** An occurrence in time, which alters the state of the system (e.g. a customer joining the queue, as described in the kiosk example above).

- **Event list:** A listing of events due to occur, in order of their time of occurrence. This is also termed a future event list (FEL).
- **Activity:** Any operation that has a duration in time of specific length and alters the state of the system is referred to as an activity. Activities often transform the state of entities (e.g., in the kiosk example above, the operation of preparing food following a customer order. Also note the resource in this example is associated with resource availability.)
- **Process:** A sequence of time ordered events, which may include several activities, is termed a process (e.g., in the above kiosk example, the operation of taking the order, preparing the food, and serving it to the customer).
- **Simulation time:** The period of time simulated by the model. Simulation time may not be the same as wall clock time (i.e., real time). Simulation time units might vary according to model requirements, therefore one simulation time tick could represent a second, a minute, hours, days, or even weeks.
- **Duration:** The difference in clock time between when the simulation started and when the simulation terminated.
- **Run time:** The total time that is, or needs to be, simulated for the period of interest.

Figure 2.3 explains the relationship of some of the above terminologies.

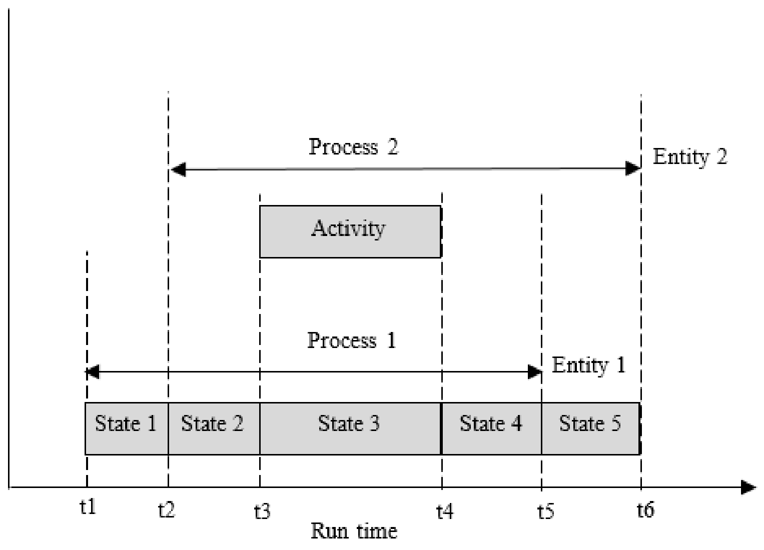


Figure 2.3: Events, Processes, Runtime, and Activity

2.2.2.1 DES Approaches

In a simulation program, execution, entities, and resources are engaged in different activities and events. The simulation programme also maintains an event list and the simulation time (this will equal real clock time if the simulation is running in real time mode). The simulation proceeds forward in time at each scheduled event in the event list. There are four different known approaches to model a discrete event simulation system (Pidd, 2006). These approaches also symbolise a distinctive "world view".

1. Event-based approach
2. Activity-scanning approach
3. Process-based approach
4. Three-Phase approach

These four methods produce programmes commonly in three level hierarchical structures, i.e., executive (control programme), operations, and detailed routines. Further details on these levels can be studied in the work of Pidd (2006). However, special attention is drawn to the second level, as it is the set of statements describing the simulation operation that make up the model. At this level the actual programme (model) logic resides and executive level, sequence these operations as the simulation proceeds. The computer receives entity interaction instructions at this level and all of the four approaches above impose their own structure at this level.

1. Event-based approach

This approach was most popular in the USA in the early years of discrete event simulation and was used in SIMSCRIPT (Markowitz et al., 1963), though later versions encouraged to use the process-based approach. This, and other reasons, affected the use of event-based simulations after 1980 (Pidd, 2006). In the event-based approach, the second level of hierarchical structure is made up sets of EVENT ROUTINES. The event routine is a collection of instructions in the computer model to execute all the logical instructions that can flow from an event. The first (executive) level must perform time scan, event identification and event execution control the operation of a simulation. This can be achieved by maintaining a proper Event List. Each entity on the list must have at least two pieces of information, i.e., the time of execution and the identity of the event. The executive level performs a two phase cycle for the duration of the simulation. These two phases are:

a. **Time Scan:** creating a list of all events due at the current simulation time, deciding the next event time from the list, and then setting the simulation clock at the next event time.

b. **Event Execution:** ensuring all due events (identified by the time scan task) are executed.

The two phase cycle continues until the simulation is over as illustrated in Figure 2.3(a). An important design consideration in this approach occurs if the simulation has many events. In this case, an appropriate list processing technique should be used to reduce the time consumed by the executive process.

2. Activity-scanning approach

The activity-scanning approach has been comparatively more popular in the UK and was used in the CSL programming language (Buxton and Laski, 1962). The focus of this approach is on the process and interaction of entities and, unlike the event approach, does not execute operations that result in a state (i.e., event) change. This approach treats each activity separately, and so can lead to run-time inefficiency. Since it treats each activity independently, the activity scan phase attempts to run every activity, regardless of the simulation condition potentially meaning that only one activity is possible. In contrast, the event-based approach focuses only on those events that need processing. These events are identified by maintaining the *Event List*. Hence, the event-based approach is faster to run, but the activity-based approach is quicker and easier to program.

In the activity-based approach *Activity* is the building block. This describes the actions that will take place once the change of state is triggered. Unlike the event-based approach, the executive level in the activity-scanning approach has only one task to perform, i.e., *Time scan*. This task represents only one responsibility, which is to identify when the next event is due. Neither time scan nor the activity scan phase attempt to identify which entity will cause the state change. Also, no attempts are made to identify the next due activity. After the time scan and activity scan the simulation clock progresses to the next time and repeats the same process until the simulation reaches its end. This activity-scanning approach is illustrated in Figure 2.4(b).

3. Process-based approach

This is a hybrid approach that combines the features of both activities-based and event-based approaches. The process-based approach to discreet event simulation modelling is the most frequently used in the world. SIMULA (Hills, 1973) and GPSS (Greenberg, 1972) are two process-based programming languages. In the event-based approach the second level is made up of event routines, therefore modellers have to analyse and identify all possible events. In the activity-based approach the second level is made up of activities, thus modellers have to identify all activities. In the process-based approach the second level is made up of processes. These processes are also an independent segment of the program. Thus, each entity is related to a specific process that starts and stops as the simulation runs but operates throughout the entire simulation runtime or life time, i.e., an entity's complete process is identified.

The executive level of the process-based approach maintains two lists of records, i.e., *Future events list* and *Current events list*. The future events list contains all events that need to be reactivated ahead of the current simulation time. The events in this list are those that are unconditionally delayed. Meanwhile, the current events list contains a record of events that need to be reactivated at the current simulation time, regardless of being conditionally or unconditionally delayed. As illustrated in Figure 2.4 (c) the executive goes through the following cycle:

a. *Future events scan:* the future events list contains the next event time.

b. *Move records:* records moved from future events list to current events list where execution time equals current simulation time.

c. *Current event scan:* Executes all the events in the current event list before proceeding to the next simulation time.

4. Three-Phase approach

Tocher (1963) first introduced the three-phased approach to combine the advantages of both event-based and activity-based simulation approaches, i.e., simplicity from activity-based approach and efficiency of the event-based approach. Initially, Tocher (1963) took the building block of the activity-based approach and extended *Activity* into two types of activities, i.e., *'B' Activities* (bound or book-keeping) and *'C' Activities* (conditional or co-operative). *'B' Activities* are executed when the schedule time is reached. *'C' Activities* depend on different classes of entities or a specific condition for their execution. Later the term "*Activities*" was removed and now they are just called Bs and Cs (Pidd, 2006).

The operations that have a predicted starting and finish time can be scheduled as if they were appointed. These are known as Bs. The Bs can be directly scheduled, therefore, the simulation executive will run them when the simulation clock is at the right time. This means that each of the Bs must be listed in the event list. All the remaining operations are regarded as Cs. These operations have no known time of occurrence, and might depend on a state change of resource or entity. The exception is operations that generate new entities in the system, which might occur at random but are classed as Bs. Figure 2.4 (item d) shows the three-phase executive operation. The reason it is called the three-phase approach is because it repeatedly executes three phases, A, B, and C.

Unlike the event-based approach the entities in the three-phased approach need to retain at least three pieces of information, i.e., *the time cell, the availability, and the next activity*. The time cell is the next time due for a state change, availability is to indicate if the entity is committed to some B, while the next activity is the B in which the entity will be engaged.

a) The A phase

The time scan is known as the A phase. During this phase the executive checks its event list to find the next event time and move the clock to that point. Then the executive will prepare a *DueNow* list. This list will contain all the operations that need to be processed at the current simulation time.

b) The B phase

Using the *DueNow* list, the executive will execute the Bs in order (there can be some priorities defined).

c) The C Phase

In this phase the executive executes remaining operations whose conditions are satisfied. The executive will repeatedly scan for other Cs in the list until no more activity is possible.

If we analyse which approach is best, then we can safely remove activity-scanning and the event-based approaches We already have established that although activity scanning is simple, it is also inefficient, while the event-based approach gets complicated once the complexity and number of entities increases, though it is faster to run. The three-phase approach overcomes the disadvantages of both these approaches. Of the remaining two approaches, three-phase and process-based are similar, as they both result from combining the advantages of previously discussed approaches.

However, in the three-phased approach the B phase is completed before the C phase, thus all dependent resources are free and all operations are addressed in a sequential process, meaning there are no changes subject to deadlock. In contrast, in the process-based approach each entity is taken through its process and may continue its process without waiting for other entity(s). This means the first entity might affect the results of the second entity. Therefore, it becomes the responsibility of the modeller to ensure that deadlocks are avoided in the process-based approach.

Nevertheless, the process-based approach does have one benefit over the three-phased approach, namely, simplicity in the design and development of models.

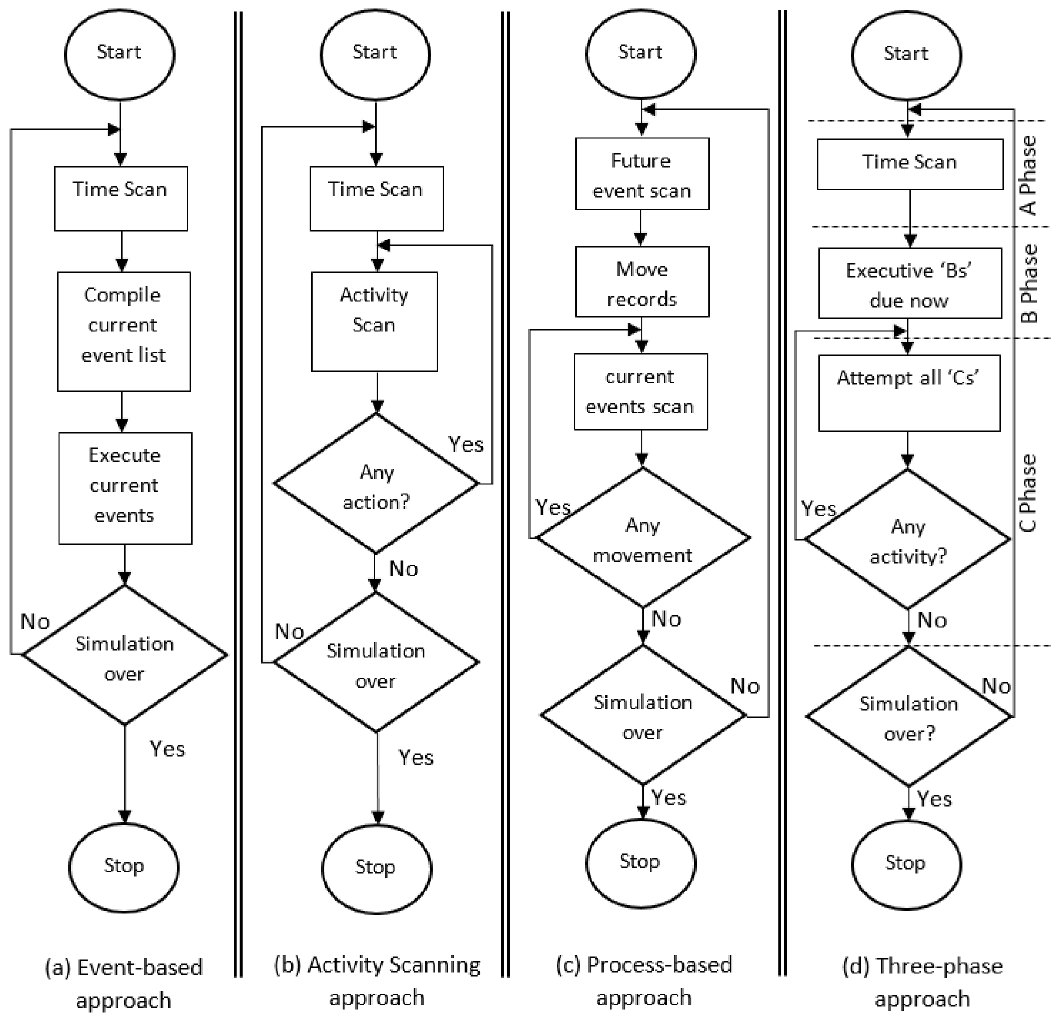


Figure 2.4: The executive of four different Simulation Approaches

2.2.3 Simulation Software

The concept of DES software goes back to the 1950s. A decade later, in the 1960s, the simulation community greatly benefited from programming languages such as FORTRAN and GSP. The first simulation languages like SIMSCRIPT, GPSS (Schriber, 1974) and SIMULA (Dahl and Nygaard, 1966) were also introduced. With the introduction of smarter computing in the 1970s, the potential of both interactive and visual simulations were highlighted by Hurrion (1976) in his PhD thesis, which resulted in the first visual interactive simulation (VIS) language in 1979, SIMAN (based on modelling framework developed by Zeigler in 1976), SEE-WHY (Fiddy et al., 1981). During the 1980s and 1990s a range of further simulation languages became available such as Simul8, WITNESS, and AnyLogic (Law and Kelton, 1999). Since then, different simulation software packages have been used across the world with different capabilities, for example, 3D displays, optimisation, databases, etc. The history of simulation programming languages was divided into the first five periods by Nancy (1995) and the sixth period by Banks (2009).

1) Period of Search: 1955 - 1960
2) Advent: 1961 - 1965
3) Period of Formation: 1966 - 1970
4) Period of Expansion: 1971 - 1978
5) Period of Consolidation and Regeneration: 1979 - 1986
6) Period of Integrated Environment (i.e. Interoperability, Composability, and Reusability): 1987 - Present

Today, the most popular type of simulation software is VIS. The example illustrated in Figure 2.5 is of an airport security system. The idea is to have a holistic visual display of how the simulation interacts rather than a logical running of the simulation model.

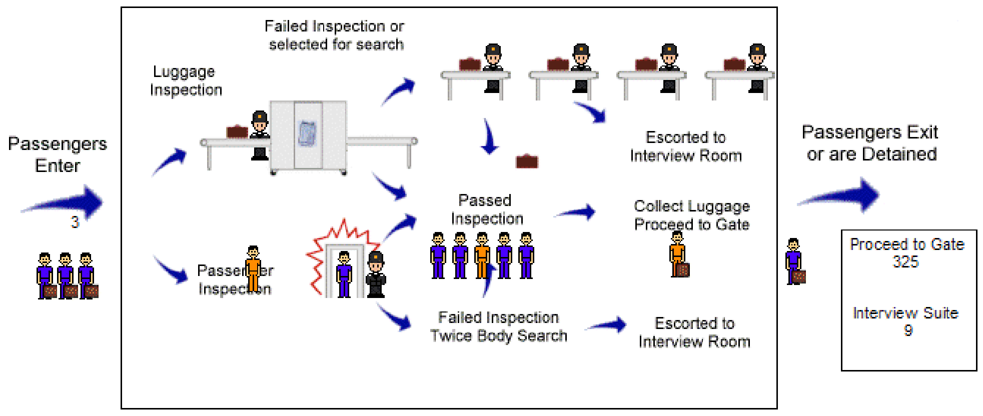
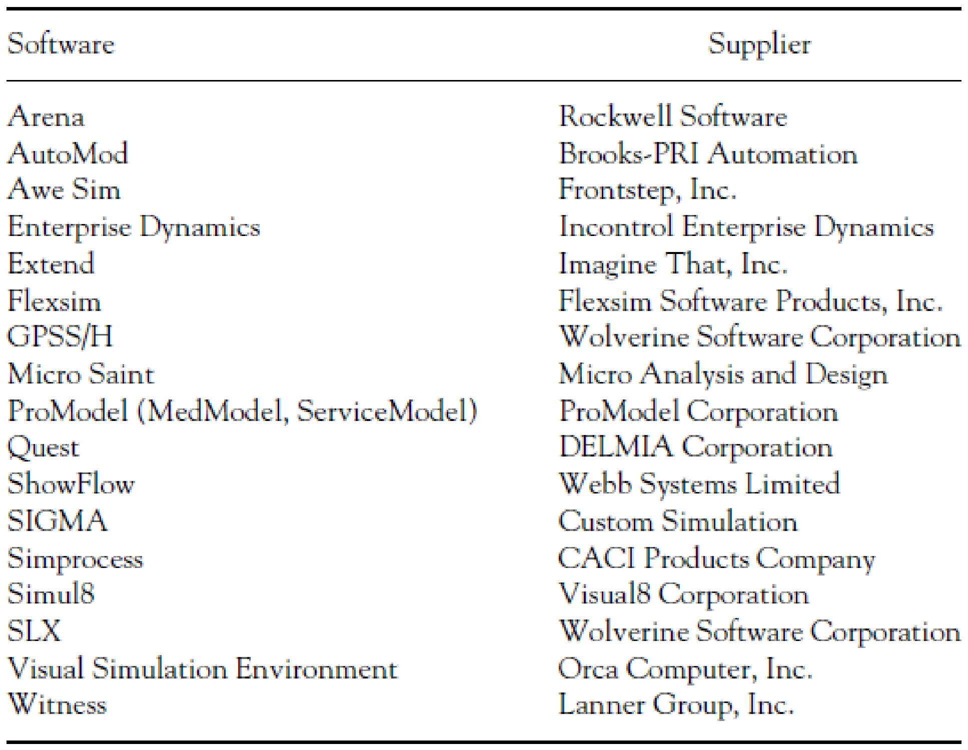

Passengers Enter
3
Luggage Inspection
Failed Inspection or selected for search
Escorted to Interview Room
Passengers Exit or are Detained
Passenger Inspection
Passed Inspection
Collect Luggage Proceed to Gate
Proceed to Gate 325
Interview Suite 9
Failed Inspection Twice Body Search
Escorted to Interview Room

Figure 2.5: Visual Interactive Simulation example for Airport Security

Further, these simulation packages provide interactive facilities such as enabling a modeller to obtain information at any point while the models are running or even stop and resume the simulation at any time. According to Stewart Robinson (a discrete event simulation practitioner and researcher), simulation modellers have three options for developing their computer simulation model, i.e., spreadsheets, programming language, and specialist software. A spreadsheet such as Excel provides a number of functions which can help develop a simulation. Programming languages such as C++, Java, or Visual Basic are suitable for developing models. The object oriented approaches used by modern programming languages are also beneficial for simulation modelling (Pidd, 1992). Finally, specialist simulation software packages are helpful in preparing a quick, visually interactive modelling simulation (VIMS) model. A short list of some specialist simulation software is given in Table 1 below. However, both Salt (1993) and Chwif et al. (2000) have argued that the computing power of a PCs (especially the memory and hardware) is not enough to address the increasing complexity of simulation models, which has given birth to the potential for distributed simulation software.



Software	Supplier
Arena	Rockwell Software
AutoMod	Brooks-PRI Automation
Awe Sim	Frontstep, Inc.
Enterprise Dynamics	Incontrol Enterprise Dynamics
Extend	Imagine That, Inc.
Flexsim	Flexsim Software Products, Inc.
GPSS/H	Wolverine Software Corporation
Micro Saint	Micro Analysis and Design
ProModel (MedModel, ServiceModel)	ProModel Corporation
Quest	DELMIA Corporation
ShowFlow	Webb Systems Limited
SIGMA	Custom Simulation
Simprocess	CACI Products Company
Simul8	Visual8 Corporation
SLX	Wolverine Software Corporation
Visual Simulation Environment	Orca Computer, Inc.
Witness	Lanner Group, Inc.

Table 1: Examples of Simulation Software packages

2.3 Modelling and Simulation

Computer based simulations are essential requirements to meet the needs of a growing world economy. More economic competition has brought bigger challenges to achieve higher goals such as greater production at lower cost. This has led to the introduction of more complex systems that can deliver more integrated logistics and more efficient production, as well as better responses to social care needs such as more effective health care systems. Therefore, the development of M&S technologies has become ever more demanding to address these critical challenges of the world of computer simulation. The M&S concept itself dates back to Shannon (1975) but greater focus and the most dramatic changes have occurred in the last few decades due to the emerging needs of new applications and improvements in computing technologies.

According to Fujimoto et al. (2017) Modelling and Simulation is defined as:

> “the discipline that comprises the development and/or use of models and simulations. Where *models* are a physical, mathematical, or otherwise logical representation of an entity, system, process or phenomenon, while *simulation* is a method for implementing a model and behaviors in executable software.”

Both “Modelling” and “Simulation”, if examined independently, have equal importance. Modelling is the representation or abstraction of real world systems and models may consist of a simplified version of such systems (Pidd, 2013). Simulation involves mimicking the behaviour of a model over time. Hence, modelling provides the abstraction level while simulation provides the implementation level (Talk, 2010). In the same paper, Talk (2010) associated modelling with conceptualisation and simulation with implementation, and defined M&S as two separate activities that are dependent on each other.

2.3.1 Simulation Studies

System engineering defines the methodologies used to conceptualise/analyse, design, and develop a system. Similarly, frameworks and methodologies are important for simulation systems to create and design system artefacts and modelling concepts that enable documentation and capture of simulation system requirements. However, it should be stressed that distributed simulation systems tend to be more complex due to the additional interoperability issues, which are discussed later in this chapter and in subsequent chapters.

There are many concepts and descriptions by different authors that outline the activities inherent in a simulation study in terms of the M&S project life cycle (Robinson, 2014). Among these authors are Shannon (1975, 1998), Hoover and Perry (1990), Gogg and Mott (1992), Ulgen et al. (1994), Law (2006), Banks et al. (2009), Sturrock (2012), and Loper (2015). As part of an early simulation study, Balci (1994) presented six processes that address simulation modelling. These processes were:

1. Problem formulation
2. Feasibility assessment of simulation
3. System and Objective definition
4. Model formulation
5. Model representation
6. Programming

Figure 2.6 (Banks et al., 2009) below defines the M&S life cycle in more detail, while Figure 2.8 (Loper, 2015) also defines the M&S life cycle, but with minor differences to the Banks et al. (2009) model.

The model from Banks et al. (2009) has 12 steps in the life cycle. The life cycle begins with *Problem formulation*. A clear formalised statement formulating the problem issue must be established either by the policy maker or by the analyst. As the system study progresses further, there can be instances where the problem statement may need to be reformulated.

The *Objectives and Project plan* are placed in the second step of the life cycle. At this stage, an evaluation should be made to decide if the simulation project is suitable for the requirements and that no alternative solution is available. A detailed plan is then prepared, including feasibility study, project duration, people involved, and the expected outcome.

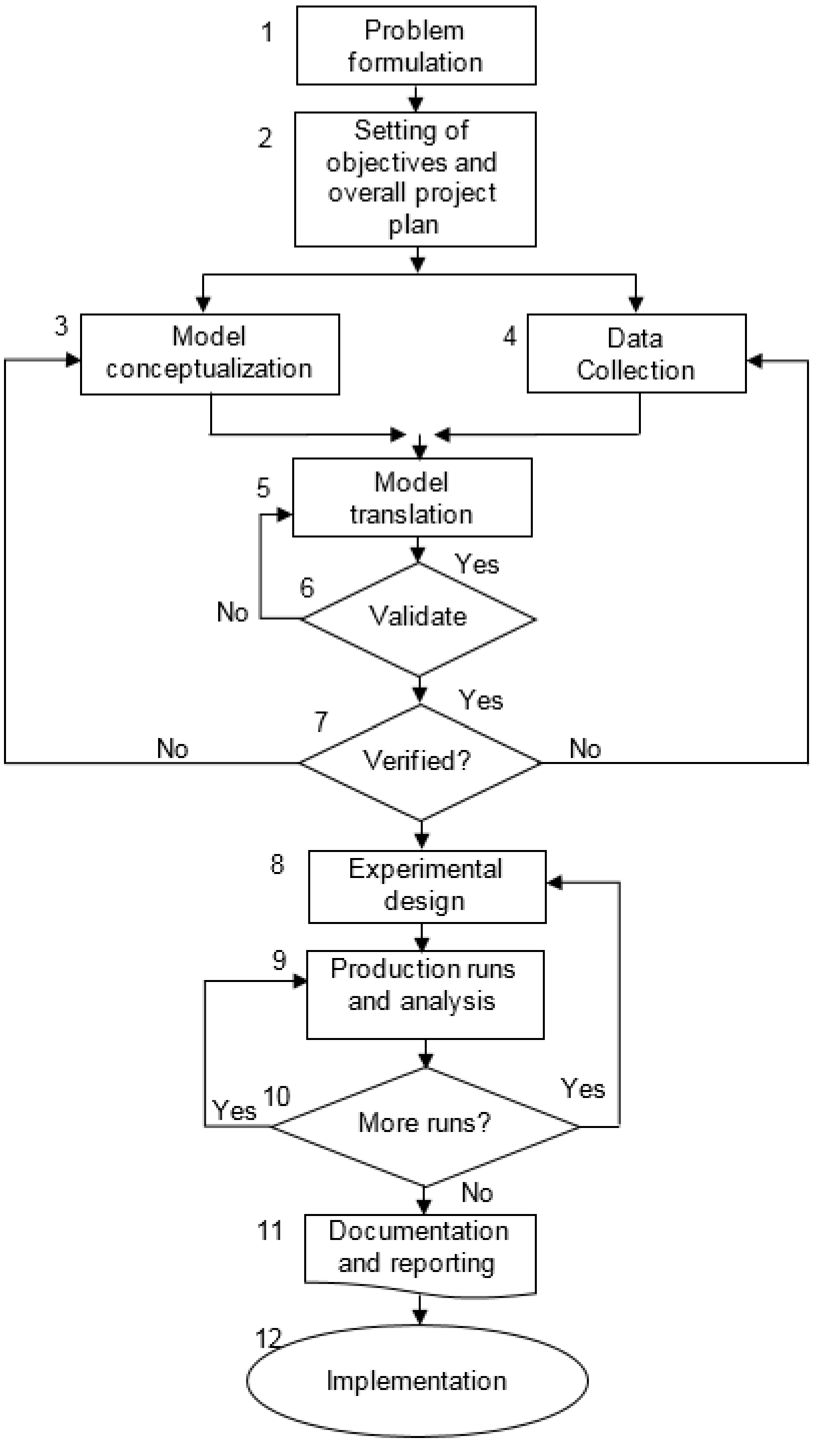


Figure 2.6: Steps in Discrete Event Simulation Study (Banks et al., 2009)

The *Model conceptualisation and Data collection* stages are divided into two by Banks et al. (2009) while others have presented them as one. Banks's model suggested that both these stages should run parallel to each other while some authors have advised completing conceptualisation before moving to data collection. Shannon (1975) identified that both conceptualisation of the models and data collection are closely connected with each other. The main reason for determining these as parallel activities is because the required data might change as the model changes in complexity. The early involvement of the end user also ensures the validity of the model and data. The data collection stage involves a substantial amount of time, therefore, it should start as early as possible. The data collected at this stage is also used late in the validation stage. This thesis will discuss the conceptual modelling stage in more detail later and in the next chapter.

The *Model translation* stage involves converting the model into computer-recognisable format via coding or programming. At this point, the modeller can decide which simulation platform will be used. The modeller might decide to use such simulation languages as Java, C++, or GPSS/H, or alternatively the modeller may employ special purpose simulation software, also known as Commercial Off-The-Shelf (COTS) simulation packages, such as Simul8 or WITNESS, for example. Using COTS simulation packages can significantly reduce development time and may offer certain flexibility, but such packages generally also come with limitations, which are discussed in more detail in Simulation Challenges.

The next two stages involve the simulation model in *Verification* and *Validation (V&V)*. The verification stage is to confirm that the translated computer program is working properly. At this stage, the program can be tested with complex case scenarios and debugging tools to analyse the output. In the validation stage, this output is then compared with the actual system behaviour. Again, a number of case scenarios are drawn up to validate the program repeatedly by comparing with data collected in the data collection stage. Once the model has achieved acceptable accuracy the process of V&V will stop.

In the next two stages, of *Experimental design* and *Production runs and analysis*, decisions regarding simulation runs are taken using cases that have been analysed and completed. At this point, it is decided how long the simulation will run, i.e., the duration of each simulation run, and also how many times each simulation should be run. The results of these simulation runs are used for evaluating the simulated system performance. They also determine if further runs are required in the next *Further runs?* at decision stage. The experimental stage is also called the "*What if*" analysis stage as illustrated in Figure 2.7 (Robinson, 2014). As the cycle runs, using some data or values as input, the results or output from running the model is analysed, then the simulation is run again with different values to monitor the change.

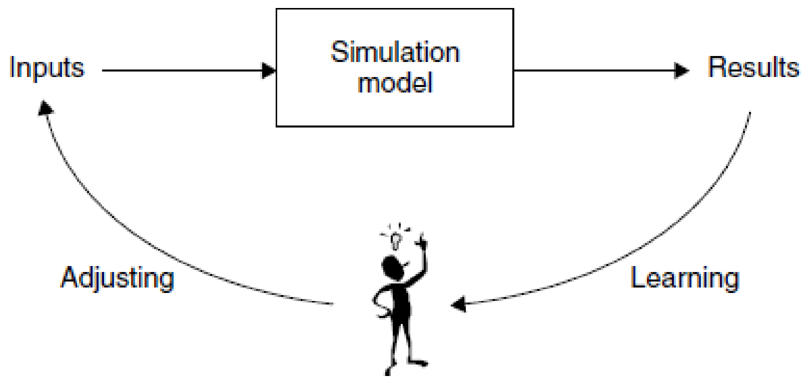


Figure 2.7: "What if" analysis with simulation (Robinson, 2014)

In the *Documentation and reporting* stage, two different documentation types are prepared. The first is Program documentation and the second is Progress documentation. The focus of program documentation is all-round re-usability, i.e., enabling the programme to be used again by documenting the functions, constraints, inputs, and outputs for use by the same or a different analyst. This documentation is also useful if the program needs modification. The progress documentation provides the written history of an individual simulation. This documentation helps users who are not directly involved in any of the related procedures or processes to understand the simulation. Ulgen et al. (1994) identified various types of progress reports such as user and maintenance manuals, logs, project book, etc.

The final stage is the *Implementation* stage. Successful implementation of the project is based on user acceptance and there is a strong relationship between acceptance and user involvement throughout this stage. Also to be noted at this stage is that if the objectives set at the second stage are not met, then the implementation could suffer, regardless of acceptance.

Banks et al. (2009) have further summarised these 12 stages into 4 phases. The first phase relates to the discovery and comprises *Problem formulation* and *Setting of Objective and Overall Design*. The second phase concerns model building and consists of *Model Conceptualization, Data Collection, Model Translation*, and *Verification and Validation*. The third phase relates to running the model and consists of *Experimental Design, Production Runs & Analysis*, and *Additional Runs*. The fourth phase of implementation consists of *Documentation & Reporting*, and *Implementation*. This four-phase approach is used by Robinson (2014) to summarise conceptual modelling and is discussed later in more detail.

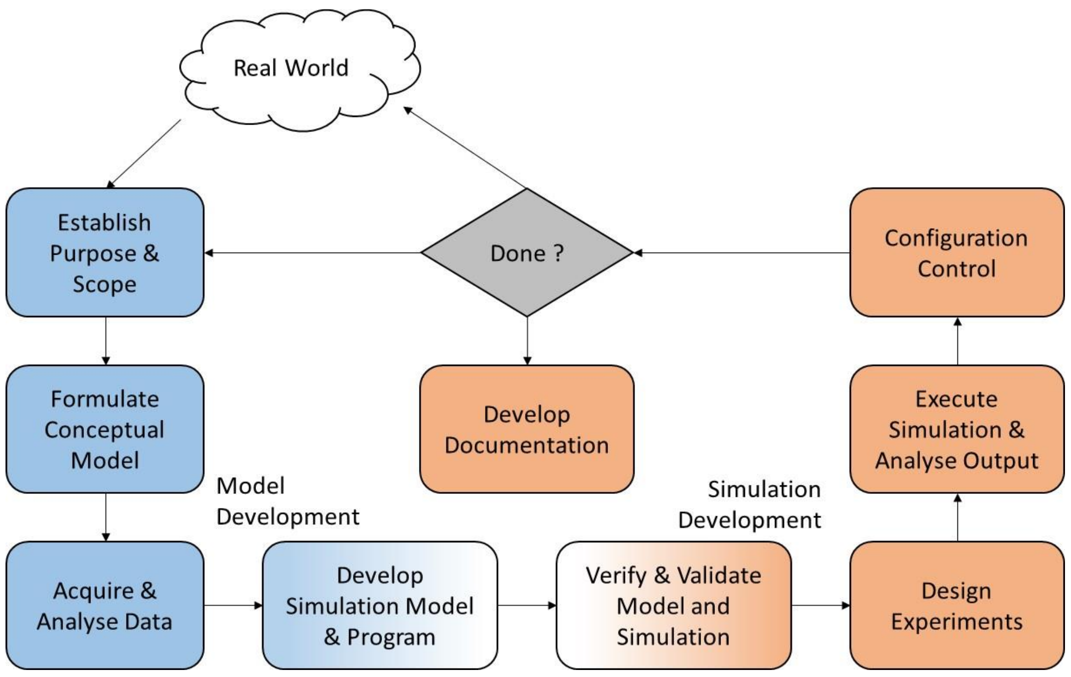

Real World
Establish Purpose & Scope
Done ?
Configuration Control
Formulate Conceptual Model
Develop Documentation
Execute Simulation & Analyse Output
Model Development
Simulation Development
Acquire & Analyse Data
Develop Simulation Model & Program
Verify & Validate Model and Simulation
Design Experiments

Figure 2.8: Modelling and Simulation Life Cycle (Loper, 2015)

The model proposed by Loper (2015), shown in Figure 2.8, is a very similar concept to that of Banks et al. (2009) but with some differences. For example, the conceptualisation and data collection stages are not parallel. On the other hand, the V&V stages are not separate stages but are performed together. *Configuration control* and *Execute simulation & analyse output* are similar to the production run and analysis stage presented in Figure 2.6. While Banks summarised the development life cycle into four phases, Loper (2015) classified the life cycle into two major activities or phases. “Model development activities” are presented by blue boxes and “Simulation development activities” are presented by orange boxes.

On the other hand, IEEE has recognised that there are many possible ways to build distributed simulations (IEEE Std. 1730-2010). The diversity of these different approaches acknowledges the desire to avoid unnecessary practices in developing such an environment. For example, the approach to developing and executing distributed training exercises can be different to the analysis oriented simulation environment. Therefore, IEEE published the standard for recommended practice for Distributed Simulation Engineering and Execution Process (DSEEP) (IEEE Std. 1730-2010). This approach for distributed simulation modelling is also acknowledged by Anagnostou and Taylor (2017).

At an abstract level, IEEE has recommended seven basic steps for developing and executing distributed simulation (IEEE Std. 1730-2010).

1. Define simulation environment objectives
2. Perform conceptual analysis
3. Design simulation environment
4. Develop simulation environment
5. Integrate and test simulation environment
6. Execute simulation
7. Analyze data and evaluate results.

Figure 2.9 describes the top-level view of the process. Further detailed process flow can be studied from the Standard. The first step is for both user and developer to agree the objectives to be accomplished by the simulation goals. Once the goals are identified the development team proceeds to the *Conceptual* modelling phase (discussed in more detail in the next section). *Design the simulation environment* involves developing a plan to meet the required functionality and reuse of models. Elaboration of a data exchange model is conducted during the *Develop simulation environment* phase. The *Integration and testing simulation environment* phase involves testing interoperability requirements. After *Execution of the simulation*, the final phase involves *Analysis and evaluation* of the received output data.

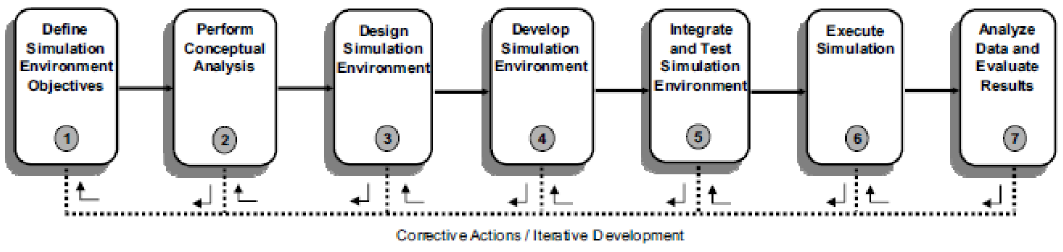

Define Simulation Environment Objectives
1
Perform Conceptual Analysis
2
Design Simulation Environment
3
Develop Simulation Environment
4
Integrate and Test Simulation Environment
5
Execute Simulation
6
Analyze Data and Evaluate Results
7
Corrective Actions / Iterative Development

Figure 2.9: Distributed Simulation Engineering and Execution Process (DSEEP) (IEEE Std. 1730-2010)

Up to now, various approaches for developing a simulation model were discussed, but the IEEE recommendation introduces the missing ingredient for this discourse, i.e., the development of a *distributed* simulation model. The purpose of Step 5 in this development life cycle *(Integrate and test simulation environment)* is to conduct verification on the integration of different parts of the simulation models and test the validity of the interoperability between these models.

2.3.2 Conceptual Modelling

In the examples of the M&S life cycle described above, Conceptual Modelling represents the first and most important stage of model design, after identifying the problem domain. Conceptual modelling was initially regarded as an abstract simplification of the real world system (Zeigler, 1976) (Pidd, 2003). Later, conceptual modelling was defined by Robinson (2008) as:

> "A non-software specific description of the computer simulation model (that will be, or has been developed), describing the objectives, inputs, outputs, content, assumptions and simplification of the model" (Robinson, 2008a).

As stated in this definition, conceptual modelling is not related to the software of a computer model, rather it creates the foundation to develop a computer model. Also, conceptual modelling describes the abstract of the real world system (including assumptions), which will be used to develop the computer simulation. The definition refers to a simulation model “that will be, or has been developed” because the modelling process is considered to be iterative and likely to change during the development life-cycle (Balci, 1994; Willemain, 1995; Robinson, 2014). Finally, the definition describes a list of deliverables, i.e., objectives, inputs, outputs, content, assumptions, and simplifications. Onggo (2007), further described these deliverables in more detail.

It is the responsibility of the system analyst or engineer to fill the gap between the conceptual model and implementation, and they need strategies to transfer knowledge between these activities (Tolk, 2010). Various authors have defined methodologies for designing a good conceptual model and for identifying what level of complexity should be used for model design (more of this is discussed by Pidd (2010)). The focus of this thesis, however, is on getting the correct level of simplification of a distributed simulation model rather than discussing the level of complexity of individual models. Therefore, further discussion will focus on the abstraction of simulation models. In the early 21st century the key elements of conceptual modelling were identified as:

- Moving from problem situations to how and what is going to be modelled
- An iterative process
- Independent of code or software
- Simplified (abstract) representation of the real world

Robinson (2001), noted that the key activities identified in simulation studies conducted by various authors were largely similar. The main differences were in the activity names and their subdivisions. Therefore, Robinson presented a simplified version of conceptual modelling as shown in Figure 2.10. The boxes in the figure represent key stages and the arrows represent the activities or process between the stages. A model is valid and considered accurate if its response is within acceptable range of its requirements (Balci and Sargent, 1984). Based on this theory, Tolk (2010) presented a similar conceptual model focusing more on validation and verification.

This approach also confirms the ideas of Banks, et al. (2009) on modelling techniques. Figure 2.11 illustrates the modelling concept proposed by Tolk (2010). If we closely compare both conceptual models, then very little difference can be found. The idea presented by Robinson (2001) focuses more on bridging the gap between real world and simulated model, while the focus of Tolk (2010) is on simulation credibility. As the validation and verification phases are present also in the Robinson (2001) theory, this will be discussed in more detail later.

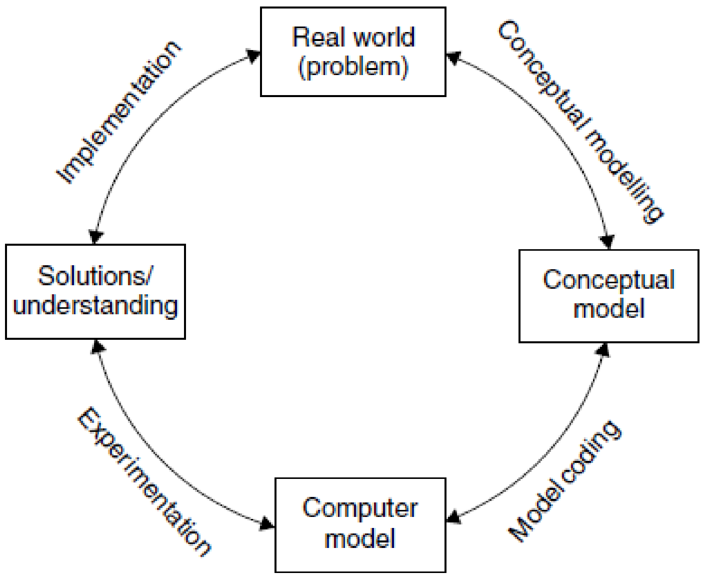


Figure 2.10: Simulation studies: Key stages and activities (Brooks and Robinson, 2001)

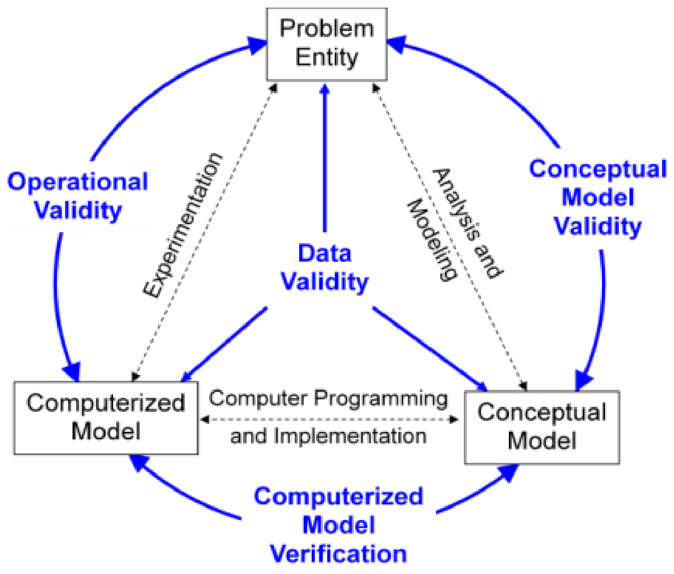


Figure 2.11: Modelling, Validation, and Verification (Tolk, 2010)

In Figure 2.10, the stages of Conceptual modelling, Model coding, Experimentation, and Implementation are defined as processes that result in the Conceptual model, Computer model, Solutions/understanding, and the start point Real world, respectively. (The Solutions/understanding stage was referenced as “Improvements/understanding” in later text by Robinson (2014)). Arrows at both ends of arcs represent the iterative nature of the model, hence the modeller can go back at any stage, while the circular representation of the diagram indicates that the process could be repeated over and over again.

Although a range of approaches have been proposed, there are no set standards for documenting DES conceptual models (Robinson, 2015). The following are some alternative examples:

- Component list (Robinson, 2014)
- Process flow diagram (Robinson, 2014)
- Activity cycle diagram (Robinson, 2014)
- Logic flow diagram (Robinson, 2014)
- List of assumptions and simplifications (Robinson, 2014)
- Unified modelling language (UML) (Richter and März, 2000)
- Petri nets (Torn, 1981)
- Condition specification (Overstreet and Nance, 1985)

2.3.3 Conceptual Modelling

Conceptulisation for simulation modelling is defined as “a set of steps and tools that guide a modeller through the development of a conceptual model” (Robinson, 2015). Another definition of conceptual modelling given by Pegden (2010) states,

> “A simulation modelling world view provides the practitioner with a framework for defining a system in sufficient detail that it can be executed to simulate the behaviour of the system. Unlike simple static descriptive tools such as Visio, IDEF, UML, etc., a simulation modelling world view must precisely define the dynamic state transitions that occur over time.

The world view must provide a definitive set of rules for advancing time and changing the state of the model."

As described in previous sections, while there are many modelling techniques, there is very little guidance available (Robinson, 2015). Recently, some authors have attempted to devise different conceptual modelling frameworks, such as "A conceptual modelling framework for manufacturing" (van der Zee, 2007) and "The ABCmod conceptual modelling framework" (Arbez and Birta, 2010). Similarly, Karagoz and Demirors (2011) have presented frameworks such as KAMA, FEDEP and BOM.

Based on the description of conceptual modelling in Figure 2.10, Robinson (2014) illustrated his conceptual modelling framework as shown in Figure 2.12. According to his framework, there are five main activities: *Understanding the problem situation*, *Determining the modelling and general project objectives*, *Identifying the model output*, *Identifying the model input*, and *Determining the model content* (Robinson, 2014).

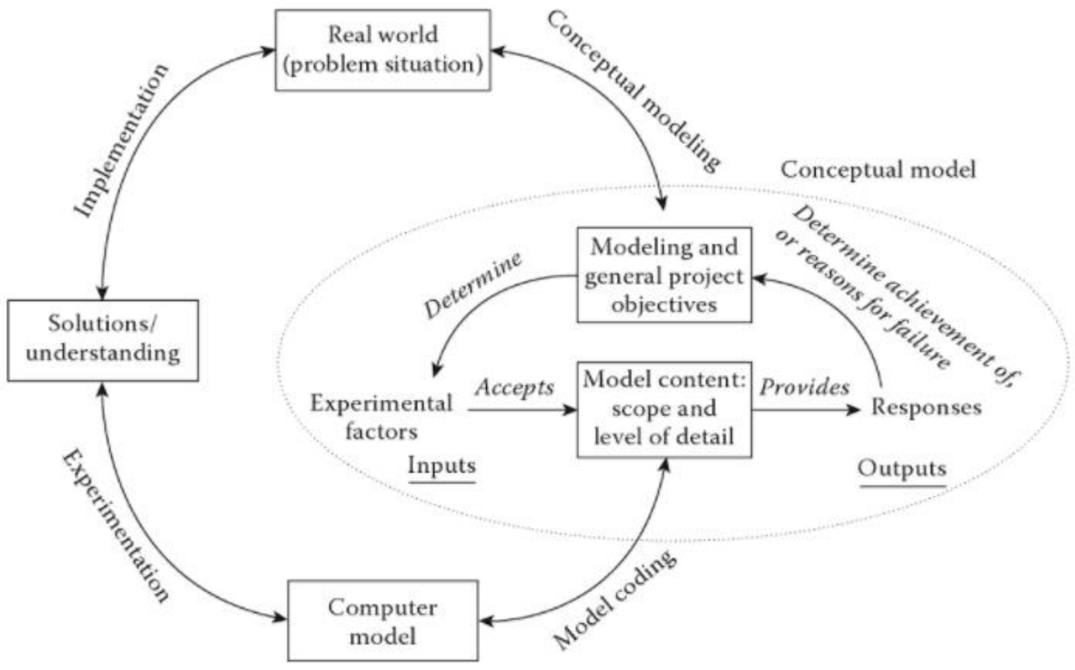


Figure 2.12: Robinson's Conceptual Modelling Framework (Robinson, 2014)

According to Robinson (2014), Understanding the *problem situation* will result in a list of modelling objectives. These objectives can be further divided into two types. First, the modelling objectives that define the reason for the modelling project. The second type is general project objectives and might include timescale, model flexibility required, visual display, run-speed, and usability. These objectives are intrinsic to the model and therefore are related to the specific model.

Different *Inputs* can result in better understanding of the problem situation by testing alternative input data. *Output* presents the responses from those different simulation runs, and determines achievement or failure of the objectives. Finally, *Model content* is determined based on the input and output. The most important thing to consider here is whether the model can accept the input and produce the desired output. The modeller must decide at this stage if a need for simulation exists or the problem could be resolved with other alternatives.

2.3.4 Artefacts of Conceptual Modelling

In 2011, Robinson simplified the key artefacts of conceptual modelling, as shown in Figure 2.13 below. On close examination of Figure 2.13, we can see that the artefacts are more or less the same as the deployment life cycles defined by other authors, with some changes to names and procedures. The key contribution is that the complete life cycle is divided into two parts (differentiated by the colours green and yellow). The top (green) section contains artefacts related to the *Problem Domain* while the bottom (yellow) section has artefacts related to the *Model domain*.

In the Problem domain there is a cloud that represents a current or future Real world problem, and the System description, which defines knowledge acquired from the problem situation. The Model domain contains the artefacts of a *Conceptual model*, as described in the previous section; *Model design*, which represents the design of the computer model; and finally, *Computer model* as the conceptual model represented by software.

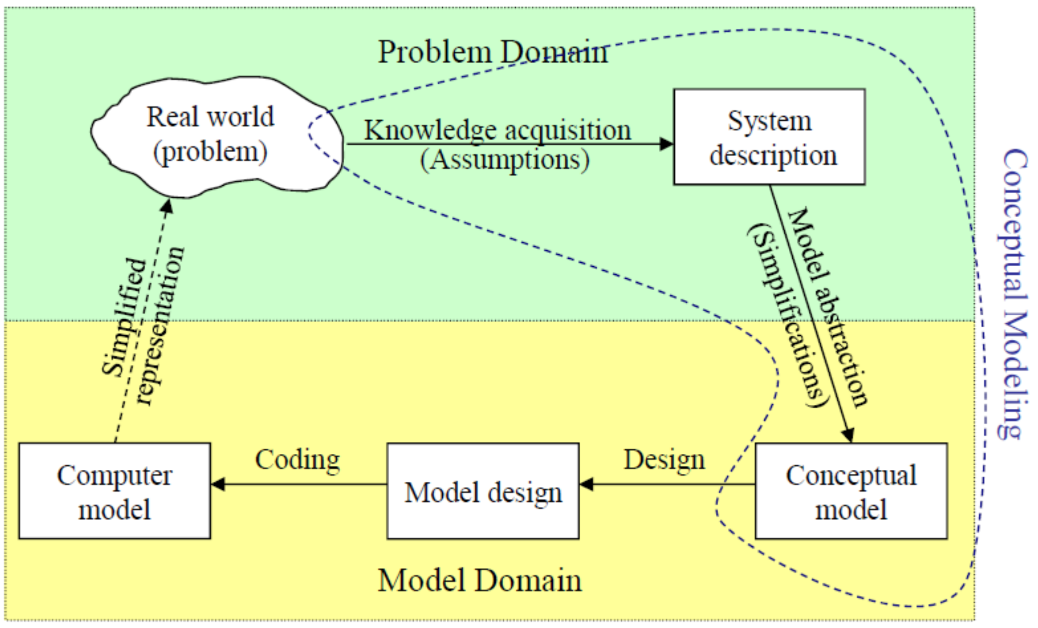


Figure 2.13: The artefacts of conceptual modelling (Robinson, 2011)

Guizzardi and Wagner (2012) have expressed a contrasting opinion. They have presented their case in terms of Software Engineering concepts and argue that conceptual models are a “solution-independent description of a problem domain”. Simply put, they believe the Conceptual model artefact belongs in the Problem domain instead of the Model domain. If this definition is divided into two parts ‘solution-independent domain’ and ‘description of the problem’, both of these are defined as independent artefacts by Robinson (2011).

On the other hand, Nance (1994) presented the idea of conceptual modelling description being separated into the Conceptual model and the Communication model. In this case, the Communication model would be the explicit representation of the Conceptual model. (Communication represents a vital role, especially in modelling distributed simulation, and it will be revisited in the next chapter.)

However, Robinson (2013) argued his case for having a Conceptual model for simulation modelling in the Model domain. The Conceptual model stage is described as how we abstract the model from our knowledge of the System description and this is the reason for separating the System description artefact and Conceptual model artefact. Nevertheless, they are also described as one by surrounding them with a dotted line to break down model complexity and understanding of the system.

The arrows in Figure 2.13 explain the flow of information. For example, knowledge acquisition from the Real world problem informs the System description. The arrows do not represent any order or flow of the modelling process, instead the modeller can return to any artefact at any point during the simulation study. These processes are defined as frequently repeating by Balci (1994), Willemain (1995) and Robinson (2014). Regardless of this, there is still some ordering to the process and information flow as the output of one artefact is fed as input to other artefacts.

Finally, the dotted arrow in Figure 2.13 between the Computer model and Real world artefacts identifies conformity between the computer model and the real world. At this stage, this correspondence relates to the accuracy of the model developed and the real world problem. This stage is also defined in more detail in Figure 2.8 as the V&V stage. During this stage, the factual and conceptual veracity of the model is correlated with the real world problem. Hence, the computer model should be appropriate and validated.

2.4 Distributed Simulation

Distributed simulation has been defined as “the distribution of the execution of a simulation program across multiple processors” (Fujimoto, 2000). Distributed simulation techniques enable the simulation of a single model by dividing it between several processors (CPU), or the simulation of multiple models by establishing connections between the different processors on which the models run (see Figure 2.14).

Beyond computer science and operational research, distributed simulation has also been used to represent portable and accessible simulated physical environments (such as in medical training) (Kneebone et al., 2010). In this thesis, the term distributed simulation identifies computer simulation models executed either over single node with different hardware specifications or with multiple nodes in a networked environment.

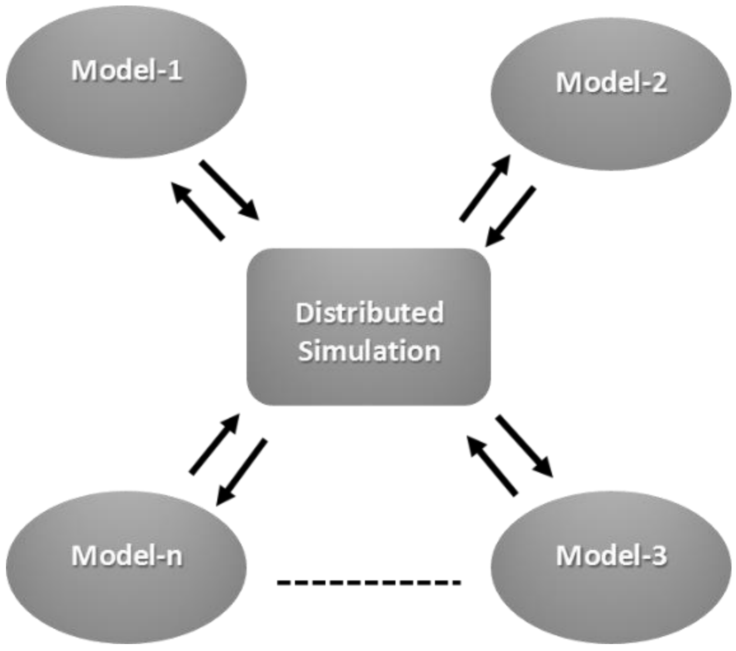


Figure 2.14: Distributed Simulation

The following are certain examples in which distributed simulation provides an advantage (Fujimoto, 1999; Fujimoto, 2003).

- Large simulations require large memory and processing capacity. By distributing execution across many machines distributed simulation provides access to the much larger resources. Distributed simulation enables the execution of combined simulation models that could not be performed with the capacity of a single computer.
- Executing a simulation program on computers that are at different geographical sites enables multiple parties based at various locations to engage in a single virtual world. For example, training exercises conducted on simulated models executing at distant locations can enable participants to interact as if on a single site, thus saving the costs and inconvenience of travel to one location.

- Distributed simulation enables the recycling (or *re-use*) of existing models, thereby delivering considerable time and cost savings. Using previously developed independent models to create a combined simulation process with other models overcomes the need to recreate established models when integrating into a single simulation environment.
- Simulation processes from different manufacturers that are designed to execute on different machines can be integrated (or *composed*) with Distributed simulation. As an example, the flight simulators for different aircraft designers can be remotely linked together to create a combined virtual environment. Rather than installing all programs on a single computer, each program can be executed on a different computer with simulation data distributed between all the computer locations.
- Distributed simulation reduces the risk associated with computer failure. A simulation running on a single processor will be brought to a halt if that processor fails. Distributed simulation involves the combination of multiple processors, therefore if one machine fails, it is possible for its work to be taken over by others, and the simulation is enabled to continue.

To achieve integration and interconnection (i.e., reusability and composability) through common architecture is still the aim of distributed simulation (Hakiri et al., 2010). However, there are a number of particular characteristics that distinguish distributed simulation from conventional sequential simulation.

Distributed simulation involves the simultaneous performance of tasks by a group of different simulators. To perform this process, simulators communicate with one another during run-time by sending and receiving messages. In contrast to sequential simulation, in which tasks are performed chronologically following a global clock, distributed simulation tasks do not follow the sequential time. It is therefore vital to ensure that accurate information is passed between models at the right time to guarantee that deadlocks are avoided and events are executed in appropriate order. This simultaneous performance of events considerably reduces overall run time and keeps communication overheads to a minimum.

In addition to the above advantages, it is possible for multiple geographically separated participants to be embedded in the Distributed Virtual Environments (DVEs) and be able to intercommunicate in the virtual local environment. Simulators from different organizations can also be integrated via coordination mechanisms, thus eliminating the costs of manually merging geographically distributed simulators.

Although distributed simulation is now used widely in military applications, its adoption in other sectors lags behind (Taylor et al., 2002a). A chief cause for this is the degree of technical expertise involved in implementing satisfactory communication protocols between distributed simulation models. However, there is a belief that standardising distributed simulation practices could significantly help to overcome this obstacle (Taylor et al., 2012).

The development of distributed simulation has resulted in the most common standard in use today, i.e., IEEE-1516-2010 High Level Architecture (HLA). This standard developed out of earlier protocols named Distributed Interactive Simulation (DIS) and Aggregate Level Simulation Protocol (ALSP), both of which originated with the US DoD.

DIS emanated from the SIMNET military combat simulation project. It was proposed as a standard method for message changing in a virtual world with the status of an IEEE standard (IEEE-1278, 1993). DIS enabled common semantics for describing and implementing many relevant concepts such as system coordination, entity description, message specification, network communication, interoperability, scalability, and latency. DIS avoided the need for central control over execution as the individual distributed applications involved managed operation by changing the status of entities and objects through the simulation process as required. Dead reckoning was employed to compare local with global status and reduce communication demands. Standard Protocol Data Units (PDUs) enabled interoperability and runtime communication between different simulators. Software such as OpenDIS and KDIS support the DIS standard.

The ALSP is a protocol that enables individual simulations to interoperate with one another using supporting software (Baker, 1999). The Mitre Corporation was responsible for developing it within the US military for use in training and analytical simulations. ALSP involves specific infrastructure software (AIS) to support distributed runtime simulation, an interface that offers generic data exchange protocols, and dedicated simulation formats. ASLP was designed as a confederated system that enables communication between federated simulation models. Its development sought to overcome DIS limitations of restriction to local area networks (LAN) and real time simulations by providing data and time management services and object ownership, which can be dynamically changed between federates during simulation execution.

This section will list some of the challenges faced by the simulation modeller while developing a distributed simulation model. At this step it is important to identify and appreciate these challenges. In later chapters there will be discussion on how to address these challenges at different stages of the system development life cycle (i.e., Conceptual modelling).

2.4.1 Interoperability

Interoperability refers to the ability to exchange data or information between two or more systems or models. The principal challenge for distributed simulation middleware is to achieve interoperability. The objective is to make software applications (simulation models) that have been developed independently not only achieve interaction through remote message passing, but also provides synchronisation so that messages can be correctly interpreted and responded to appropriately. Effective communication is the key factor between two models, therefore understanding the issues related to interoperability between interacting simulations is important (Taylor et al., 2012b).

Leal et al. (2017) have also raised the importance of interoperability in enterprise business by suggesting that to remain competitive big enterprise businesses need to be connected with other companies. Businesses are becoming more interactive and are acting more like a networked enterprise, according to Jagdev et al. (2001).

Communication and collaboration between such companies, to exchange and share competencies, have thus increased the need for more interoperable systems.

Interoperability has been in focus for many years. Harkrider and Lunceford (1999) stated that "technical integration of systems is necessary but not sufficient". The observations of Dahmann et al. (1998) drew attention to the distinction between technical interoperability and substantive interoperability. Petty (2002) then extended technical interoperability requirements further by introducing communication, hardware, and a protocol layer. Page et al. (2004) further decomposed the implementation layers for interoperability both in the technical layer (for integratability) and modelling layer (for composability). Meanwhile, Fujimoto et al. (2017) has observed that "Achieving interoperability across these modelling approaches is an open problem".

The difficulty arises when independent simulation packages may have been developed by different teams using different semantics and synchronisation algorithms. It is true that application modellers can develop expertise with familiar tools, but are challenged by adapting to other application methods (Al-Zoubi and Gabriel, 2011).

A principal user of distributed simulation technology is the defence sector where distributed simulation is used particularly to create virtual training environments that connect separate parties. In this sector, the development of HLA middleware protocols is predominant in providing generic, interoperable simulation architecture that is capable of reuse. However, these protocols still lack the ease of interoperability of plug-and-play, composability, and scalability. (Al-Zoubi and Gabriel, 2011).

Distributed simulation interoperability means that simulation models can interact effectively, both technically and substantively. The US DoD Modelling & Simulation Coordination Office (M&SCO) characterises technical interoperability as the ability of federate models to physically connect and exchange data. The requirement is for common standards, compatible interfaces, and coordinated data structures.

The stated elements comprising technical interoperability are:

- Hardware compatibility
- Standards compatibility
- Time management coordination
- Coordinated use of RTI services
- Control over security issues (Anon, 2011)

Substantive interoperability refers to the ability of interconnected federates to provide accurate, appropriate, and reliable simulated representations that meet simulation objectives.

The stated elements comprising substantive interoperability are:

- Logical interaction between the entities of distributed federates
- Temporal resolution
- Spatial resolution
- Coherent relationships between the components of the real-world physical environment. (Anon, 2011)

2.4.2 Reusability

The software engineering community recognised the economic benefits of reusability many years ago. Reese and Wyatt (1987) defined the reuse of software as "isolation, selection, maintenance and utilization of existing software artefacts in the development of new systems". McIlroy's law states that software reuse "reduces cycle time and increases productivity and quality" (Endres and Rombach, 2003).

Petty et al. (2010) defined reuse as, "Using a previously developed asset again, either for the purpose for which it was originally developed or for a new purpose or in a new context". Balci et al. (2011) defined reusability as "the degree to which an artefact, method, or strategy is capable of being used again or repeatedly". In the above definitions, and in the simulation modelling world, an asset or an artefact is termed as a software component that can implement a model or part of a model. Reusability offers the opportunity for the community to "stand on the shoulders of giants", according to Fujimoto et al. (2017).

Various approaches to reusability and its elements have been identified. Pidd (2002) describes the different forms of software reuse in terms of a spectrum, as shown in Figure 2.15. In this diagram, the frequency arrow shows the prevalence of software reuse in terms of different software elements. It shows that reuse is more common with smaller elements of software rather than with complete models. The arrow indicating complexity supports this principle by showing that the more complex, dedicated, or complete the software the less it is likely to be adopted for reuse. The conclusion observed is that the simpler the reuse operation, such as with code scavenging, the more it will be adopted, whereas complex full models are rarely chosen for reuse.

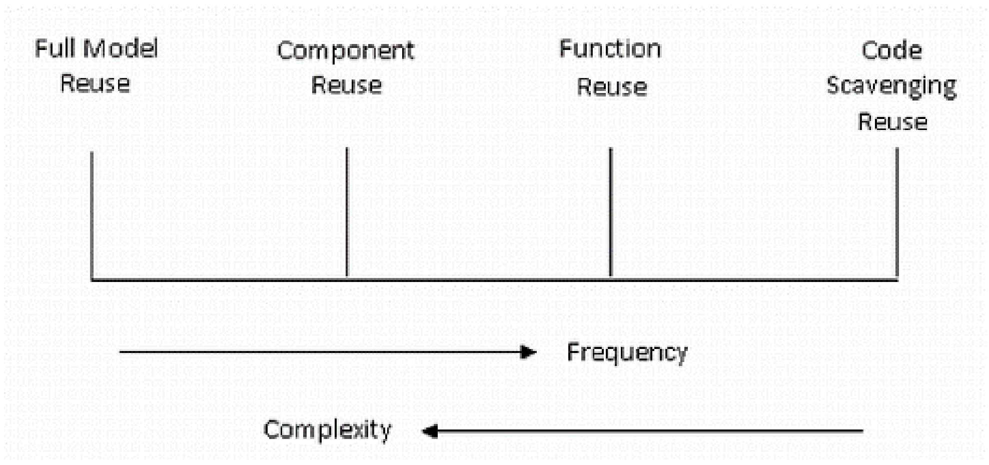


Figure 2.15: Reuse Spectrum (Pidd, 2002)

Pidd’s Reuse Spectrum indicates that Code Scavenging is the most common form of reuse adopted by programmers and designers. Cutting and pasting previously used code is a simple way to achieve simple objectives. It is often the way that programmers and modellers initially learn their skills. It is natural to turn to working code to address new problems, but such code scavenging is mostly practised by the creators of the original code.

Function Reuse involves more than simple code and indicates some form of module service. This mainly indiucates reuse of previously defined modules or a built-in functionality. Component Reuse suggests the reuse of built-in components designed for an individual application that uses domain specific language or libraries.

Components, in this sense, are also described as building blocks (Verbraeck et al., 2002; Oses et al., 2003).

Full Model Reuse means reuse of a complete model in unaltered form. A complete application model clearly involves the most complexity and thus generally the most specificity. The opportunity to reuse full models provides the greatest economic benefit by saving development time. However, this is only practical if the need is to solve the same problem for which the model was designed. Most design challenges will not involve the same system or an identical problem, therefore, reusing a complete model for an alternative purpose requires further modification to the new model (Fishwick, 1995). Therefore, appraisal of the precision and credibility of the reused model will be required (Balci, 1997).

Paul and Taylor (2002) classified the reuse of simulation models from the viewpoint of the commercial package modeller. Three groups were identified: basic modelling component reuse; reuse of subsystem models; and reuse of similar models. One concern highlighted was the importance of trust verification arising from reuse of simulation models, noting this could be a time-consuming and costly process. Today, technologies such as web-enabled simulations provide added support for the reuse of models, enabling better analysis of functions in model building.

Robinson et al. (2004) discussed the advantages and difficulties of the various levels of model reuse, drawing parallels with object-oriented design and programming in terms of modularity and reusability. A significant conclusion was that purposeful consideration of reusability at the beginning of a project is more likely to deliver reusable simulations, with such careful recycling having a beneficial impact on time and costs. However, there is generally insufficient impulse for a modeller to follow this path.

Balci et al. (2008) viewed the potential for reuse of simulation models from the higher perspective of conceptual modelling. Given that a conceptual model is a preliminary and simplified form of a domain-specific, real-world process, it was argued that conceptual models offer the most opportunity for reuse. This study concluded that the higher the level of abstraction the greater the availability for reuse, regardless of the ultimate method of simulation implementation. However, the study focus was on static rather than dynamic simulation models.

Garro and Falcone (2015) have observed the difficulties presented in reusing simulation models that are potentially available for reuse. The problem is due to the lack of structures enabling simulation models built on different platforms to interoperate; lack of support for execution on distributed infrastructures being a chronic difficulty. Similarly, Taylor et al. (2015) identified that "specialised knowledge" would be required for some models to be reused.

Awareness of the disadvantages resulting from this lack of support has produced considerable research into methods, techniques and models that can enable reuse and interoperability of simulation models in the distributed computing environment. Two of the most significant developments are FMI (Functional Mock-up Interface) (Fmi-standard, 2017) and HLA (IEEE Std. 1516-2010). HLA is discussed later in this chapter in more detail. Currently, the focus of the industry is on implementation standardisation to address reusability (Tolk et al., 2007).

2.4.2.1 Reusability and Interoperability

Effective communication between models has been identified as one of the barriers to achieving reusability (Fujimoto et al., 2017). Such effective communication, in these terms, refers to achieving interoperability across models. Interoperability, reusability, and composability are critical factors in the design of large-scale complex models, and appropriate implementation of these factors could result in a significant decrease in time and costs of development. This research is a step forward in addressing reuse problems by standardising the known interoperability issues affecting reusability.

2.4.3 Composability

The term composability was first used in the mid-1990s in the context of US defence industry simulation during the Composable Behavioural Technologies (CBT) project. The expression was reinforced in the late 1990s during the Joint

Simulation System (JSIMS) project (Page and Opper, 1999). Composability is described by various authors in different ways.

Lunceford and Harkrider defined composability as, "The ability to create, configure, initialize, test, and validate an exercise by logically assembling a unique simulation execution from a pool of reusable system components in order to meet a specific set of objectives" (1999).

Kasputis defined composability as, "The ability to compose models across a variety of application domains, levels of resolution, and time scales" (Kasputis, 2000).

Petty and Weisel (2003) recommended the following definition in their article on the theory of composability, which was later appended by Davis and Robert (2004): "Composability is the capability to select and assemble simulation components in various combinations into valid simulation systems to satisfy specific user requirements, meaningfully".

Dahmann et al., (1998) drew attention to a distinction between technical interoperability and substantive interoperability. As stated earlier, Lunceford and Harkrider (1999) viewed technical integration of systems as "necessary but not sufficient". Petty (2002) extended the scope of technical interoperability by introducing the communication, hardware, and protocol layers. While, Page et al. (2004) further decomposed interoperability implementation layers into the technical layer for integratability and the modelling layer for composability.

Later, numerous methodologies and issues concerning composability were published, which triggered further research on developing standards and frameworks for composability (Mahmood, 2013). Such standards and frameworks include HLA, ALSP, DIS, Extensible M&S Framework (XMSF), Base Object Model (BOM), Open Simulation Architecture (OSA), and Discrete Event System Specification (DEVS). Nevertheless, composability still remains a big simulation challenge (Taylor et al. 2015). As Fujimoto et al. (2017) have stated, "We face serious technical challenges in achieving reusability, composability, and adaptability for developing simulation models". Also, composability and reusability issues have a serious impact in the development of ontologies due to inherent relationship mapping (Taylor et al., 2015).

Simulation models are developed for specific purposes, therefore they depend on context sensitive assumptions (Robert et al., 2004). Therefore, composing simulation models can be more difficult than composing simple software components. Also, in most cases the composition of separate models cannot be valid (Weisel et al., 2003). Currently, the focus on composability by the industry is on achieving implementation standardisations (Tolk et al., 2007). There are certain challenges on higher levels that also need attention, together with the focus on implementation questions (Tolk et al., 2007). Modellers are more focused on IEEE 1516 and IEEE 1278, which define implementation standardisation, and pay less attention to conceptual modelling; but to ensure interoperability between systems the modelling level should not be neglected, according to Tolk et al., (2007) and Robinson (2014). Thus to achieve composability in simulation modelling it is important that both technical and modelling interoperability are standardised.

A formal theory, Semantic Composability Theory (SCT), was established at the Virginia Modeling, Analysis & Simulation Center by Petty and Weisel (Petty and Weisel, 2004). Figure 2.16 below defines an SCT model as a computable function:

$$y = f(x)$$

Where f is a finite procedure, which relates to each input and a unique output.

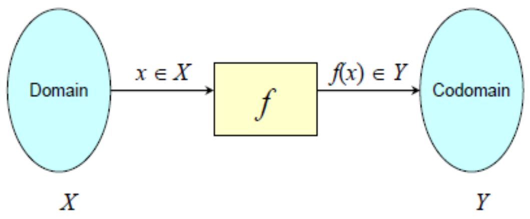


Figure 2.16: Computable function (Petty and Weisel, 2004)

The theory of composability remains under development where comprehensive, mature, and coherent theories are still being researched to identify how to compose a model and verify the composition (Fujimoto et al., 2017).

2.4.3.1 Composability and Reusability

In view of the discussion above it should be pointed out that composability and reusability are not the same (Balci et al., 2011). Composability is where large, complex and sophisticated models are constructed from existing functional-specific components. Reusability is based on an isolated, generic functional design in which components may be used in other models. Thus, composable components can achieve reusability, but reusable components might not always fulfil the composability objectives. For example, development of a large model may be composed of smaller functionally specific components, which cannot be reused in different models because of their specific input and output. On the other hand, reusability is achieved when a component is scalable, adaptable, and general. These reusable components have a more abstract design and are intended to be generic for use in different models. Figure 2.17 below explains the relationship between levels of reusability and composability.

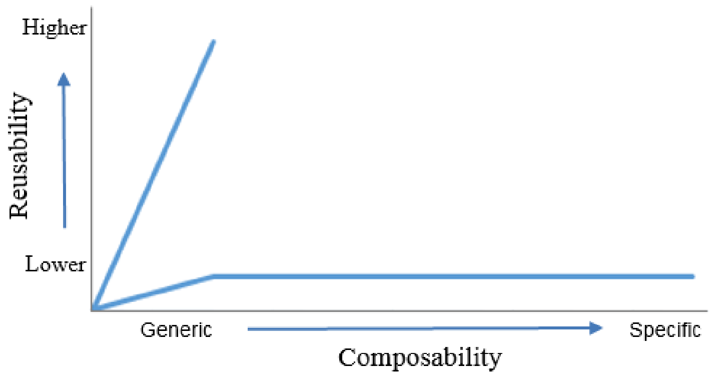


Figure 2.17: Generic vs. Specific component design

2.4.3.2 Composability and Interoperability

IEEE defines interoperability as "The ability of two or more systems or components to exchange information and to use the information that has been exchanged". A more thorough definition of composability and interoperability uses the following terms:

"Composability contends with the alignment of issues on the modeling level. The underlying models are purposeful abstractions of reality used for the conceptualization being implemented by the resulting systems; whereas

Interoperability contends with the software and implementation details of interoperations; this includes exchange of data elements via interfaces, the use of middleware, mapping to common information exchange models…" (Page et al., 2004)

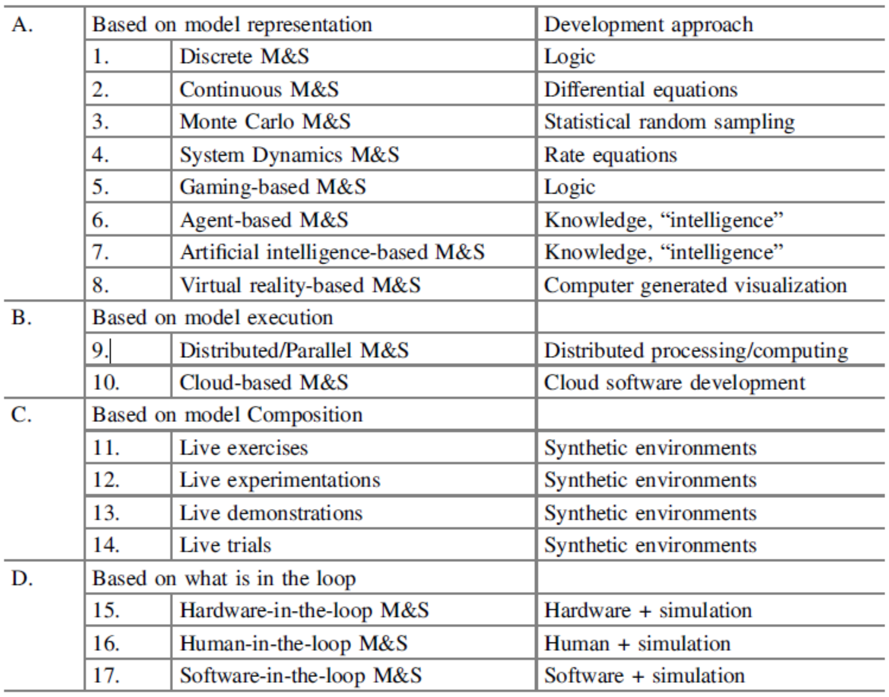
One of the more recent challenges highlighted is to achieve composability, reusability, and interoperability by focusing on "effective communication", as raised by Fujimoto (2017). The concept of interoperability is to achieve composability at a technical level, where interoperability is not just dealing with interconnection between various models (as agreed both syntactically and semantically) but also it should be standardised to make it composable for the model, while the composability is achieved at the component or model level.

Petty and Weisel (2003) described two types of composability: syntactic composability and semantic composability. Syntactic composability relates to composable components developed with standard message-passing algorithms for effective communication; semantic composability refers to whether the composed simulation comprises meaningful composed models and the combined computation is valid (Petty and Weisel, 2003).

2.4.4 Multi-formalism

Certain formalism in system specification is expected when building a model for simulation. Formalism is accepted as a convenient way to express models representing particular system classes or that address specific problems. Zeigler refers to formalism as "the types of modelling styles, such as continuous or discrete, that modellers can use to build system models" (Zeigler et al., 2000).

However, in reality, real-world phenomena are often unable to be modelled into a single type of formalism. Complex systems may comprise several different components and structures, which do not fit into a description using a single overall formalism. For example, an automated traffic control system may require modelling as both a discrete and a continuous simulation (Zeigler et al., 2000). Table 2 below identifies M&S area types and the development approaches used. Each area listed in the table has its own methodology and characteristics. This causes considerable technical challenges for reusability, composability, and adaptability. For example, one model could be developed using System Dynamics while another might be developed using an Agent-based approach.



A.	Based on model representation		Development approach
	1.	Discrete M&S	Logic
	2.	Continuous M&S	Differential equations
	3.	Monte Carlo M&S	Statistical random sampling
	4.	System Dynamics M&S	Rate equations
	5.	Gaming-based M&S	Logic
	6.	Agent-based M&S	Knowledge, "intelligence"
	7.	Artificial intelligence-based M&S	Knowledge, "intelligence"
	8.	Virtual reality-based M&S	Computer generated visualization
B.	Based on model execution		
	9.	Distributed/Parallel M&S	Distributed processing/computing
	10.	Cloud-based M&S	Cloud software development
C.	Based on model Composition		
	11.	Live exercises	Synthetic environments
	12.	Live experimentations	Synthetic environments
	13.	Live demonstrations	Synthetic environments
	14.	Live trials	Synthetic environments
D.	Based on what is in the loop		
	15.	Hardware-in-the-loop M&S	Hardware + simulation
	16.	Human-in-the-loop M&S	Human + simulation
	17.	Software-in-the-loop M&S	Software + simulation

Table 2: Modelling and Simulation area types (Balci, 2016)

To address this, either a tool is required supporting multi-formalism modelling or each component must be modelled with a tool that supports the most appropriate formalism in each individual case (Vangheluwe and De Lara, 2002). In distributed model-based problem solving, heterogeneous tools are used to build integrated sub-models representing different system components that are modelled in heterogeneous formalisms. In this context, Anagnostou (2014) presented a paper for hybrid model reusability between Discrete M&S and Agent-based M&S techniques, in this work, she discussed how different types of interoperability (covered in later chapters) are addressed to achieve multi-formalisation.

2.5 High Level Architecture

2.5.1 Introduction

The history of HLA can be traced back to the emergence of DSS in the late 1980s. The development of distributed simulation techniques by the US DoD followed Congress approval in 1988 of the Simulation Network program (SIMNET) managed by the Defence Advanced Research Project Agency (DARPA) (Hollenbach, 2009). A conference in April 1989, named Interactive Networked Simulation for Training, identified several other initiatives in the same field that were then being developed by industry. The conference grew into DIS workshops based upon the SIMNET project (Hakiri et al., 2010; Calvin et al., 1993). In 1996, the development of HLA standards led DIS to transform into a more functional structure called the Simulation Interoperability Standards Organisation or SISO.

During the 1990s, DARPA also employed the Mitre Corporation to study DIS principles. This research expanded into the linking of US Army and US Air Force simulations (air warfare simulation). The success of this project led to the development of the ALSP, with the focus centred on interoperability issues. Later, research under both DIS and ALSP was merged to propose the new standard Higher Level Architecture or HLA.

In 1991, with the concept becoming more widely acknowledged, the Defence Modelling and Simulation Office (DMSO) was created to encourage joint interoperability and reuse in the Modelling & Simulation industry (Hollenbach, 2009). By 1994, DIS was recognised for providing huge support to interoperability for DoD simulations (Hollenbach, 2009). In early 1995, the Architecture Management Group (AMG) was commissioned by the Executive Council for Modelling and Simulation (EXCIMS) under DoD to develop the HLA. The Modelling and Simulation Master Plan (DoD 5000.59-P) was provided by AMG with a remit for "a common high-level simulation architecture to facilitate the interoperability of all types of simulations among themselves and with C4I systems, as well as to facilitate the reuse of M&S components" (DMSO, 1995).

The baseline HLA version 1.0 was announced in September 1996 and approved by AMG. In the years that followed, HLA compliance was tested and reviewed and two new releases of HLA Version 1.3 were introduced as IEEE Standard 1516-2000 and more recently IEEE Standard 1516-2010.

The objective of the HLA is to enable simulation systems to work in conjunction with each other. As a generic framework, the HLA focus is primarily on interoperability rather than on specific domains. The aim is for generalised operability solutions that are applicable to many different applications. Close control over individual simulations within a distributed simulation enables HLA use in a wide variety of domains, with assurance of the vigorous but flexible approach typically applied in the military training domain (Rainey et al., 2015).

The HLA structure defines simulation entities that operate together as a *federation*. Each HLA-compliant simulation entity within a federation is named as a *federate*. A federate may be a simulation model, a data recording system, or an interfacing to a live system such as radar. The simulation entities comprising the federation typically operate together as a process providing a service or solution. Each of these federates is able to communicate via run-time infrastructure (RTI) middleware using a common Object Model Template (OMT). In HLA, an *object* is a dataset passed between federates within the federation. An *event* is an interactive communication between federates (Rainey et al., 2015). Interfacing between federate models is implemented by the RTI software in accordance with HLA specifications. Typically, a C++ library or an alternative language library is provided by the RTI to enable federate data exchange.

2.5.2 HLA Specification

The integrated architecture of the HLA is intended to offer a common architecture for Modelling & Simulation (IEEE Std. 1561-2010). Distributed simulation involves the combination of several different simulation models. In some cases, the simulation models already exist, while in other cases, they may need to be developed and integrated. In normal circumstances, huge modifications to existing models can be required for integration into a new simulation, provided the source code is available. Alternatively, all simulation components have to be redesigned from scratch. In other words, simulation modelling suffers from two underdeveloped properties: *interoperability* and *reusability*. The concept of the HLA is to provide a framework that addresses these properties.

Both reusability and interoperability are closely related to each other. Reusability implies that simulation models can be used again in other simulation models. Interoperability means that either reused or newly developed simulation models can interoperate with each other effectively without needing to change the code.

The HLA standard uses the distinct terms *federate* to denote each participating simulation model and *federation* to denote the complete combined simulation inclusive of its other components (as illustrated in Figure 2.18). To facilitate communication between federates (and to offer common architecture for distributed M&S) the simulation modeller must use standards to achieve portability, reliability, and flexibility, and to develop simulation models recognisable to other modellers. It is important to note that HLA only supplies a standard with no implementation.

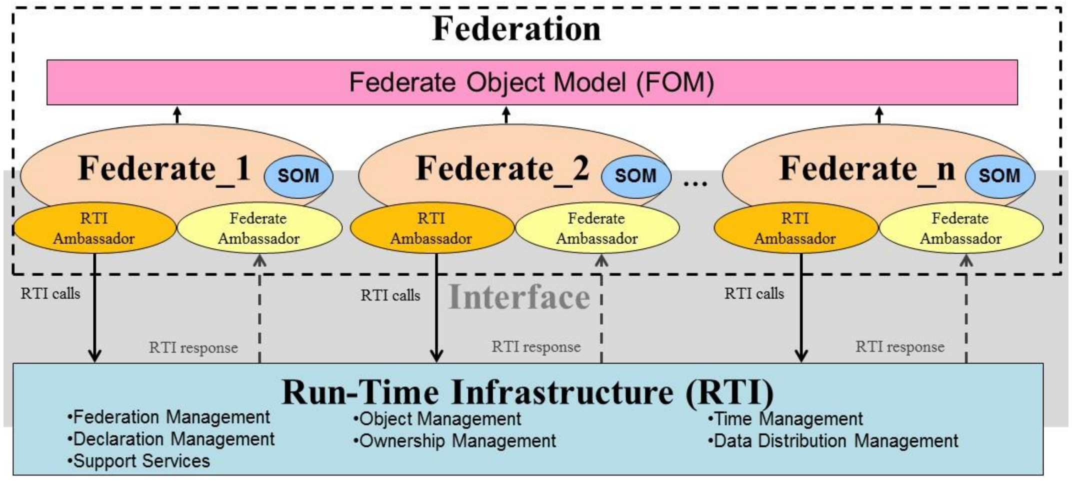


Figure 2.18: HLA Overview (Nouman et al., 2013)

The HLA Standard provides an Application Programme Interface (API) specification for communication between federates known as the Runtime Infrastructure (RTI). These specifications are defined in the HLA Federate Interface Specification document (contained in IEEE Std. 1516.1-2010). The RTI functions similar to an operating system for the distributed simulation environment by providing federate interaction and federation management services. The services provided by the RTI to support federate communication within a federation are illustrated in Figure 2.18.

The HLA therefore consists of the following three services.

a) HLA Framework and Rules
b) HLA Object Model Template (OMT)
c) HLA Federate Interface Specification

2.5.2.1 HLA Framework and Rules

HLA Framework and Rules are defined in IEEE 1516-2010 Standard specification. This specification presents a set of 10 rules split into two groups. The first five of these rules apply to federations and the remaining five apply to federates. Together, these rules ensure successful interaction for both federates and federation and list the responsibilities of each. These rules are compulsory for an HLA-compliant distributed simulation environment.

The five federation rules enforce compliance of the federation object model (FOM) with the HLA OMT and require location of simulation objects, ownership of information (data), and communication procedures between federates to be in compliance with HLA interface specifications. The five rules that apply to federates enforce use of the Simulation Object Model (SOM) in accordance with the HLA OMT and local time management.

The **federation rules** as defined by IEEE Standard 1516-2010 are as follows:

1) Federations shall have an HLA FOM documented in accordance with the HLA OMT.
2) In a federation, all simulation-associated object instance representation shall be in the federates, not in the RTI.
3) During a federation execution, all exchange of FOM data among joined federates shall occur via the RTI.

4) During a federation execution, joined federates shall interact with the RTI in accordance with the HLA interface specification.
5) During a federation execution, an instance attribute shall be owned by at most one joined federate at any given time.

The **federate rules** as defined by IEEE Standard 1516-2010 are as follows:

6) Federates shall have an HLA SOM, documented in accordance with the HLA OMT.
7) Federates shall be able to update and/or reflect any instance attributes and send and/or receive interactions, as specified in their SOMs.
8) Federates shall be able to transfer and/or accept ownership of instance attributes dynamically during a federation execution, as specified in their SOMs.
9) Federates shall be able to vary the conditions (e.g., thresholds) under which they provide updates of instance attributes, as specified in their SOMs.
10) Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation.

2.5.2.2 HLA Object Model Template (OMT)

The HLA OMT is defined in the IEEE Standard 1516.2-2010 specification. Effective communication between distributed simulation models, requires a standard protocol that enables models to understand the language of each other. This can be achieved either by adding individual translators (creating extra overhead) or by agreeing a single standard. The OMT specification adopts the latter approach by providing a common template for specifying data exchange between federation members and facilitating the design of common tool sets.

Although HLA provides standardised specifications for the Object Model, compliance differs from the definition of object models generally expected in the object-oriented world. The object-oriented concept (Straßburger, 2001) usually treats the object as abstract, with definition relying on associated attributes and performance method data. In HLA, object abstraction is limited to focus on the internal data exchange between federates rather than on the simulated object of the model. Similarly, the data presented in HLA object models are identified by attributes gained from direct access by – and updated by – other federates. Further differences involve object interaction whereby HLA federates interact by sending and responding to attribute and event messages instead of direct interaction. Responsibility for updating an object lies not with the object itself, but with updates passed between the distributed federates. All relevant interfaces for HLA object models are defined by either object classes or interaction classes (Straßburger, 2001). These classes are used to formulate the Federation Object Model (FOM) and Simulation Object Model (SOM).

i. **Federation Object Model (FOM)**

   The goal of the FOM is to create a specification for data exchange among all participating federates in a common and standardised format (IEEE Std. 1516.2-2010). It comprises an inventory for all objects and interactions relevant to the federation. An FOM module could be created for a single federation or could be formulated from multiple FOM modules.

ii. **Simulation Object Model (SOM)**

   The purpose of the SOM is to specify all the information required by a federate from other federates in the federation and all the information that is to be provided by a federate to other federates. Essentially, the SOM provides the structure for sending and receiving information to and from federates participating in the federation.

2.5.2.3 HLA Federate Interface Specification

The interface specification (HLA-IS) is abstract; its aim is to standardise an approach to persistent problems in a distributed application. The HLA-IS explains how federates will interact with the federation, and ultimately with one another, using language independence. It provides services and a communication mechanism to ensure information exchange between federates, usually implemented within the RTI (Straßburger, 2001). The interface has seven basic services and an eighth grouping of support services. The eight support services describe the interface between the RTI and federates and also between software services and the RTI. These services are further discussed in more detail in the next section.

- *Federation Management service* offer basic functions required to create, control, and delete a federation execution.
- *Declaration Management Service* include publication, subscription, and supporting control functions. Federates which produce Object Class Attributes or Interaction must declare exactly what they are able to publish.
- *Object Management Service* involves registration, updates, and dynamic transfer of the object, attributes, and interactions.
- *Ownership Management Service* allows federates to transfer the responsibility for updating and deleting object instances and also transfer the ownership of object/attributes.
- *Time Management Service* focus on the mechanics required to establish synchronization between distributed entities at runtime and ensure the delivery of messages.
- *Data Distributed Management (DDM) Service* provides a flexible and efficient routing of data among federates for isolating publishers and subscribers.
- *Management Object Model (MOM) Service* provides access to RTI operating information during federation execution.
- *Support Services* are the eighth additional service utilised by the federation like converting handlers, getting update rates values, setting advisory switches, getting federate names, etc.

2.5.3 Runtime Infrastructure (RTI)

HLA federations are constructed on bus topology. This is an aspect of the RTI, which provides underlying software enabling HLA services to be applied to the federation. Simulation federates are enabled to publish their data and output to other federates and to subscribe to data from other federates. The RTI also provides the federation with essential management functions by hosting the FOM in XML file format, which determines the kind of information data that can be transferred within the federation (Rainey et al., 2015).

Appropriate choice of RTI software can be critical to the satisfactory federation operation in terms of both performance and cost. RTI software is available from many sources including government, universities, commercial companies, and open source projects. A vital issue for consideration at the start of a project is whether there is a need for the RTI to be compatible not only with all federates in a federation, but also with other federations. As the scope of simulation requirements increases, it is not exceptional to find the need for interoperability with other federation simulations. To enable intercommunication between federations that operate on different RTI brands often requires use of a connecting bridge. This can increase cost and latency and reduce stability. Considering an RTI type already in use by potential connecting federations can avoid such problems (Rainey et al., 2015). Useful benchmarks for evaluating RTIs have been drawn up by Knight et al. (2002**).** The active standard from SISO that provides guidance for building HLA federations is the DSEEP (IEEE Std. 1730-2010).

Runtime Infrastructure (RTI) works as an HLA Proxy middleware to integrate two or more models or systems. This layer must be simple and flexible enough to allow easy integration with different application programs. The HLA goal of reusability requires this middleware to be adaptable for different platforms (e.g., Windows, Linux, etc.) and for alternative programming languages (e.g., C++, Java, etc.). Therefore, RTI maintains a common interface enabling the application to communicate, which works as an abstract layer between RTI and the application, as illustrated in Figure 2.19. This common interface is implemented as RTI Ambassador and Federate Ambassador (see Figure 2.18). The FOM abstraction module provides an interface for objects defined in the FOM document as attributes and interaction classes. These objects are transparently access the internal RTI database values to read or write. The FOM module is separated from the rest of the RTI common interface because it will be different for each model, i.e., the interaction objects might differ.

The HLA implementation of RTI generally adopts TCP/IP, UDP/IP, or HTTP protocols functioning at technical level. The FOM and SOM are at the core of HLA federations and correspond with the HLA OMT specification. Since the OMT does not specify the semantics of data exchange, FOM and SOM can only function at the syntactic level (Wang et al., 2009).

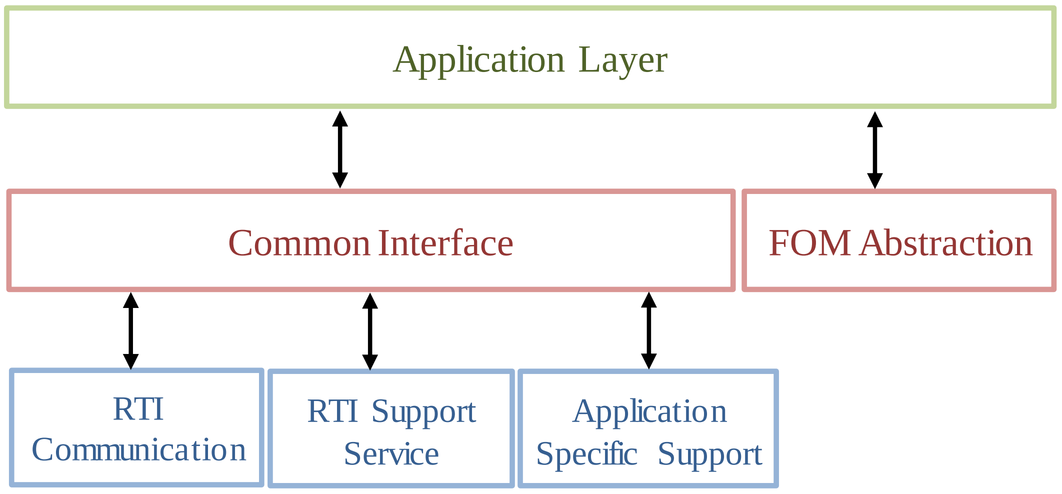


Figure 2.19: RTI middleware communication architecture

The RTI provides the following seven support services as illustrated in Figure 2.18 and Figure 2.19.

a) Federation management

“Federation management refers to the creation, dynamic control, modification, and deletion of a federation execution” (IEEE Std. 1516.1, 2010).

The RTI service requires that for a federate to undertake any interactions, including joining a federation, it must first perform a *Connect* to the RTI. Similarly, once a federate resigns from a federation, with no intent to rejoin or conduct further executions within the federation, it must perform a *Disconnect* from the RTI (IEEE Std. 1516.1, 2010).

RTI-initiated services are invoked on a federate by callbacks, which are initiated according to the specific programming language chosen for use. However, the HLA governs when callbacks can be invoked and defines this with two callback methods. The IMMEDIATE callback model requires immediate action by the RTI to invoke callbacks when available (provided they are not disabled by the *Disable Callbacks* service). The alternative EVOKED callback model requires callbacks to be actioned only if the federate has called the *Evoke Callback/Evoke Multiple Callbacks* services (provided this is not disabled by the *Disable Callbacks* service). The RTI is able to implement either one or both callback methods. Callback models are a specified argument of the *Connect* service (IEEE Std. 1516.1, 2010).

Figure 2.20 below illustrates the basic relationship between a federate and the RTI during participation in a federation execution. Federates may join an existing federation and resign from it in any order determined by the federation user. The arrows in the illustration indicate the invoking of HLA service groups (but do not strictly represent service ordering requirements). The RTI can allow a single application to join a federation as multiple federates, and a single application can also participate in multiple federations as a single federate or as multiple federates (IEEE Std. 1516.1, 2010).

b) Declaration management

DM services are required for federates that have joined a federation to declare their intentions. It is necessary for joined federates to first invoke appropriate DM services before being able to generate information such as registering object instances, sending interactions, or updating attribute values. In order to receive information, joined federates must declare this intention using either DM services or DDM (Data Distribution Management) services. A federate may use just one of these services exclusively or use both services (IEEE Std. 1516.1, 2010).

c) Object management

The object management group of HLA services controls the "registration, modification, and deletion of object instances and the sending and receipt of interactions" (IEEE Std. 1516.1, 2010).

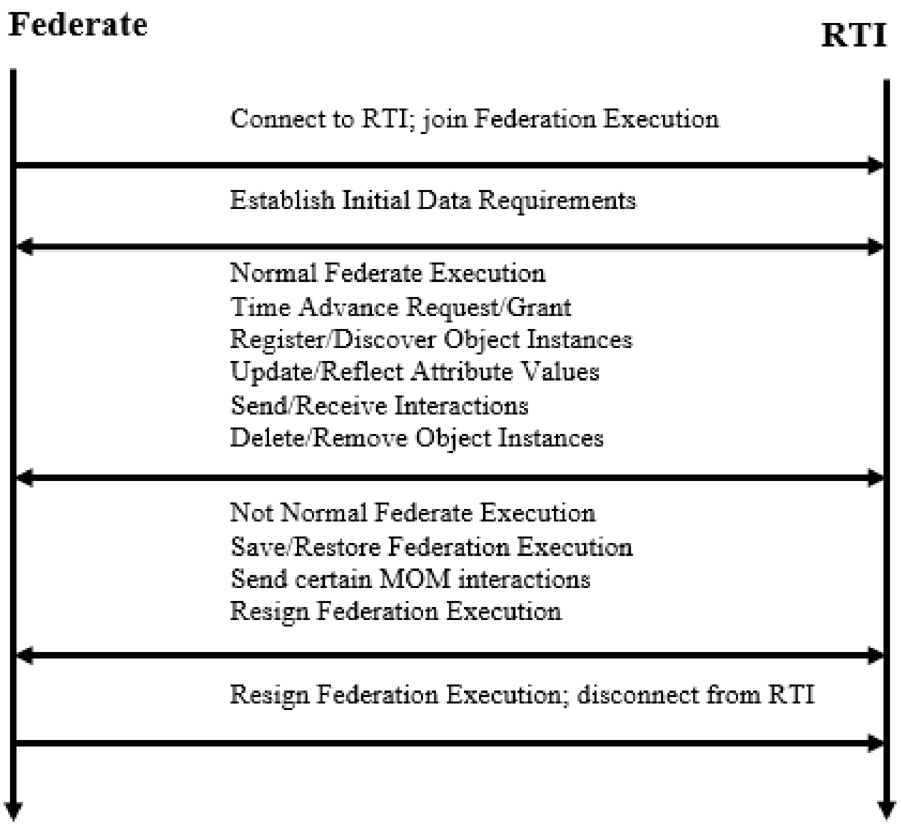

Federate
RTI
Connect to RTI; join Federation Execution
Establish Initial Data Requirements
Normal Federate Execution
Time Advance Request/Grant
Register/Discover Object Instances
Update/Reflect Attribute Values
Send/Receive Interactions
Delete/Remove Object Instances
Not Normal Federate Execution
Save/Restore Federation Execution
Send certain MOM interactions
Resign Federation Execution
Resign Federation Execution; disconnect from RTI

Figure 2.20: Basic relationship between federate and the RTI

d) Ownership management

Federates and the RTI use ownership management to transfer ownership of instance attributes between federates. Transferring object ownership between the joined federates help model management in the federation. (IEEE Std. 1516.1, 2010).

e) Time management

Time management services and associated mechanisms allow the orderly delivery of messages during execution of the federation. It is through these mechanisms that messages issued by federates are enabled to be sent and received by other federates within the federation in a consistent manner.

The time system in an executing federation is represented by points along the HLA time axis. Association with this time system can refer both to a federate itself and to certain of its activities, but each is referenced separately.

The time value of a federate in relation to the HLA time axis is described as federates' logical time (IEEE Std. 1516.1, 2010). The activities of a federate within the federation in association with the HLA time axis are denoted with timestamps. Although both time measures use the same datatype, they are denoted differently to distinguish between time values given for the federate itself (logical) and time values given for its activities (timestamped).

During federation execution, federates can advance along the HLA time axis, although such logical time advancement may or may not be restrained by the relative progress in logical time of other federates.

In managing time to advance each federate along the HLA time axis, it is essential to coordinate with object management services so that information passing and task execution are conducted in a causally correct and ordered manner (IEEE Std. 1516.1, 2010).

f) Data distribution management (DDM)

Use of DDM services enables federates to reduce overhead time by avoiding the sending and receiving of irrelevant data (IEEE Std. 1516.1, 2010). DM services facilitate by stipulating information on relevant data at the class attribute level. DDM services, on the other hand, also provide added refinement to data requirements at the instance attribute level and the specific interaction level (IEEE Std. 1516.1, 2010).

Both sending and receiving federates can use DDM services to place bounds on the relevance of interactions or instance attribute update data to be sent or received. This can be expressed by providing user-defined dimensions to create space for relevant communications. This is defined by upper and lower limits specified by both sending and receiving federates. The overlap between these limits, bounds the space for relevant communications. At the communication layer, it is the RTI that takes responsibility for recognizing irrelevant data and preventing its delivery (IEEE Std. 1516.1, 2010).

g) Management object model (MOM)

A federation requires monitoring and control of its elements from within for proper functioning of its execution. The MOM enables access by federates to RTI operating information during federation execution. Using these facilities, federates can obtain information about, and control the operation of, the federation by influencing RTI functioning and that of individual federates.

Access to and exchange of RTI information elements present in the MOM are achieved by the same methods federates use to exchange information between themselves. This involves using predefined HLA constructs, objects, and interactions. The MOM also uses the OMT format (IEEE Std. 1516.2-2010) and syntax to define these control and information elements.

The relevant, accessible activities of the RTI concern publishing object classes; registering object instances and updating their attribute values; subscribing to and receiving interactions; and publishing and sending interactions. A federate with responsibility for controlling a federation execution can subscribe to some or all of the object classes, reflect the updates, publish and send some interactions, and subscribe to and receive other interactions.

All MOM information elements (e.g., object and interaction classes, attributes, and parameters) are contained in the FOM Document Data (FDD) file. They are predefined and cannot be revised. However, a federation is not compelled to use all, or any, of these standard classes or information elements. MOM definitions can be extended by further subclasses, class attributes, and parameters for use by federates, but such new elements cannot be directly acted upon by the RTI (IEEE Std. 1516.1, 2010). Figure 2.18 above illustrates an HLA IEEE 1561 Standard implementation. The HLA standard requires modellers to use a standard application programming interface (API), i.e., “The RTI” for inter-federate communication.

h) Support services

Various miscellaneous services are available to enable federates to implement such actions as:

— Performing name-to-handle and handle-to-name transformations

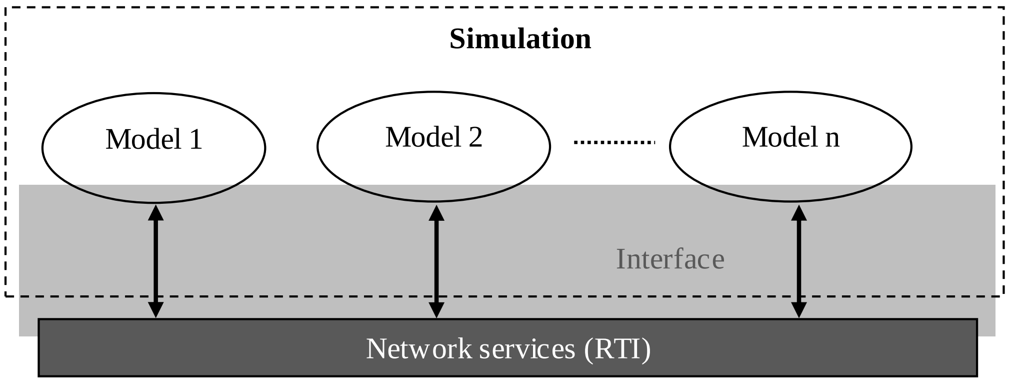
— Setting advisory switches

— Manipulating regions

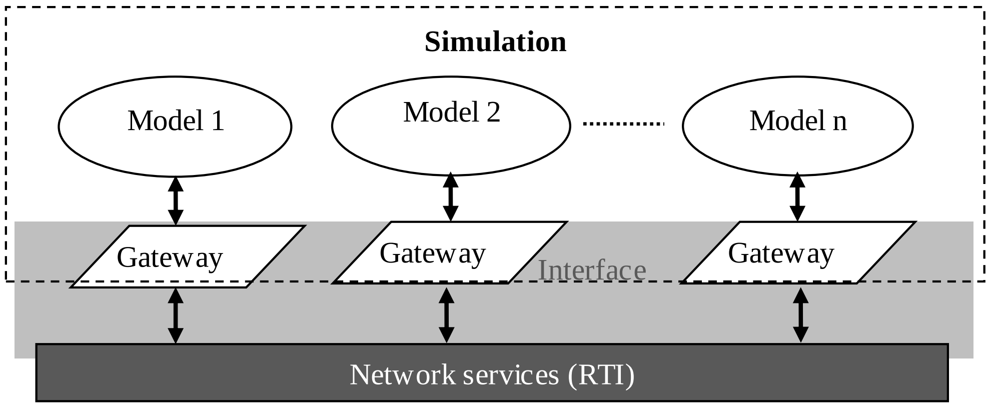
— Getting update rates values.

There are three standard distributed simulation architectures that are most widely used in the Modelling & Simulation Industry (IEEE Std. 1516, 2010). These are HLA defined by IEEE 1516, DIS defined by the IEEE 1278, and the Test and Training Enabling Architecture (TENA). All three standards face various interoperability issues, whether simulations are run in similar architectural environments or in multi-architecture environments.

There are two strategies that could be used to address these interoperability and reusability issues. The first (Figure 2.21a) is to ensure that all participating simulation models are created in the same language, which would conform to the chosen architecture. The second is to introduce a translator for participating application programmes. The former solution is easy to implement and test, and it may be suitable for small scale projects, but it is not a practical approach for large scale projects involving many participating models. Adopting a single language also seriously discourages reusability. Therefore, the latter solution is being addressed by researchers and developers with the introduction of gateways or middleware to bridge communication barriers (Figure 2.21b). This solution does support reusability; however, it introduces an overhead to the simulation. While focus has been placed on developing the interactive features of the HLA, another objective has been to create a common strategy by which HLA federations can be built. This strategy has been presented at the IEEE DSEEP (IEEE Std. 1730, 2013) (IEEE Std. 1730.1, 2013). DSEEP defines seven steps of the development process but recommends that the methodology should not be treated as prescriptive, but as a guide that can be tailored to individual circumstances. The DSEEP process follows the basic gateway/middleware concepts discussed above as strategies for addressing the model interoperability problem. According to DMAO (IEEE Std. 1730.1, 2013), DSEEP currently presents one of the good practices suitable for use by the HLA community for building HLA-compliant federates and federations.



(a) Distributed simulation environment block diagram



(b) Gateway configured distributed simulation environment

Figure 2.21: Strategies of interoperability and reusability

2.6 Distributed Simulation Interoperability Models

Single simulation models are most often created using COTS simulation packages (CSPs). Due to the variety of CSPs available from various vendors, creating a distributed simulation from models designed with different CSPs presents problems of interoperability. Such a distributed simulation requires each participating model and its CSP to interoperate with all other participating models and their CSPs.

Comparing the functionality of each approach is problematic, because of different methodologies used in creating and implementing individual simulation models. Furthermore, end-users of CSPs tend to be averse to technical specifics and CSP vendors are often confused by the complexity of interoperability techniques. Therefore, the SISO COTS Simulation Package Interoperability Product Development Group (CSPI PDG) has sought to introduce common approaches to the problems of distributed simulation interoperability. Part of this objective has involved identifying and defining a set of "IRMs" that can provide a common frame of reference when addressing solutions and help practitioners and vendors to assess particular interoperability approaches (SISO-STD-006-2010).

Antecedence to the CSPI PDG began with the UK EPSRC1 industry-focused project GROUPSIM, which identified the need for common standards to address the interoperability problems of CSPs. This led, in 2002, to the creation of the COTS Simulation Package Interoperability Forum (CSPIF), an international consortium of practitioners, CSP vendors, and researchers. In 2005, SISO recognised the importance of this work by endorsing the then newly-formed CSPIF Product Development Group (PDG). Laying the foundations for the PDG project of creating CSP common interoperation standards involved workshops, extensive research publication, and platform presentations such as at the Winter Simulation Conference (WSC), the UK ORS Simulation Workshop, the ASIM Dedicated Conference on Production and Logistics, and the Germany HLA Forum, together with continued researcher, practitioner, and vendor consultation. The products of this initial work are published as a set of IRMs proposed as tools to assist the discrete-event distributed simulation community with model development (SISO-STD-006-2010).

While a distributed simulation can comprise models created by various types of software products (CSP, Gots, custom-built, etc.), the current HLA-IRM standards are based on simulation models produced by CSPs. In the present description, therefore, a distributed simulation or federation comprises a set of CSPs and their models.

Figure 2.22 represents two models/CSPs, i.e., federates, running on separate computers. Federate F1 comprises Model M1 and COTS Simulation Package CSP1 and Federate F2 comprises Model M2 and COTS Simulation Package CSP2. For a distributed simulation, time synchronized data is exchanged between each model/CSP federate over a network (indicated by the bold double-headed arrow). To enable both Federate F1 and Federate F2 to send and receive information in an agreed time synchronised format, both must agree on common data description and communicate by the same method.

Different protocols are needed for common activities such as the sharing of information resources and the "passing" of entities. Generally, entity passing from one model to another, treats departure and arrival as a single concurrent scheduled event. Such an event is achieved by sending a timestamped event message from one federate to another federate. Messages regarding shared resources follow a different protocol. The release of a resource or the arrival of an entity at a queue requires the relevant CSP to decide on the availability of a workstation to process the entity. The changed state of a shared resource requires a timestamped communication protocol to inform and update information on the shared resource.

The research conducted by the CSPI PDG investigated the type of protocols required for simulations to meet typical real world scenario demands. The added complexities of developing workable distributed simulations to meet such demands are the challenges that IRMs seek to simplify. In a distributed simulation system, participating simulations must be able to interoperate (Mustafee and Taylor, 2006). To understand the numerous problems modellers face in achieving interoperation between interacting simulations it is of crucial importance to define these issues. (Taylor et al., 2012b).

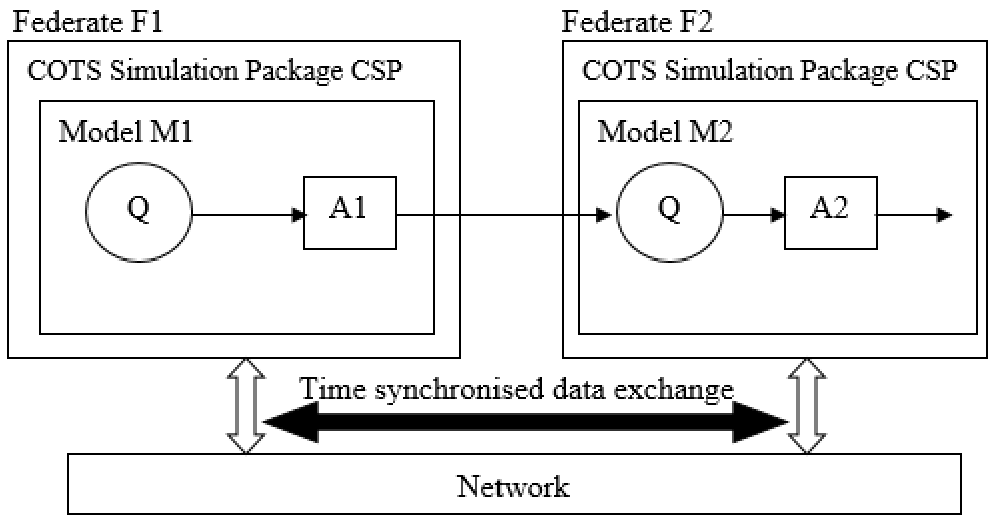

Federate F1
COTS Simulation Package CSP
Model M1
Q
A1
Federate F2
COTS Simulation Package CSP
Model M2
Q
A2
Time synchronised data exchange
Network

Figure 2.22: Interoperability Problem

2.6.1 Interoperability Reference Models (IRMs)

The purpose of the CSPI PDG, as formed in 2004 and supported by SISO, was to format distributed simulation data exchange specifications in a generic manner to complement the HLA OMT. The PDG initially identified six IRM types, each representing a particular type of data exchange.

According to SISO-STD-006-2010, IRMs represent identified interoperability problem types in simplest form and are divided into several subcategories of problem type. The simple form of each model is intended to present a generic example understood by vendors, simulation users, and technology solution providers. Each IRM generally relates to the interfacing between two or more interoperating models. Reference to time synchronisation is specified only as appropriate. IRMs represent real model/CSPs by using standard model elements that can be related to the elements of specific CSPs. All IRMs are designed to be composable to enable use of several IRMs when addressing certain problems.

As illustration, military distributed simulations are often used for cooperative combat training that requires interoperability between models developed in different countries. Although many militaries use HLA standards and common programming languages to develop models, ambiguities in the HLA result in no single approach to design, use of the RTI, or general compatibility. Research by

Serna et al. (2010) therefore proposed applying the common standards of IRM types to the particular requirements of military command and control (C2) applications.

Interoperability between different military C2 systems involves sharing three types of data: *Object-Item, Plan and Order,* and *Action*. These data types are defined by National Atlantic Treaty Organisation's (NATO) Multilateral Interoperability Programme (MIP-NATO) in a common data exchange standard - Joint Consultation, Command and Control Information Exchange Data Model (JC3IEDM). Serna et al. (2010) found the following equivalence with IRMs:

Object-Item can be a Resource or an Entity and equated with IRM types A, B, and D.

Action was an event to change a system state and was equivalent to IRM type C.

Plan and Order (did not equate with any IRM type discussed later).

Serna et al. (2010) chose to place focus on equating the methodology of military entity transfer (i.e., Object-Item) with IRM types A.1, A.2, and A.3.

Three methods of appropriate time management, interfacing were chosen from several available HLA standards for the definition of a proposed military IRM Type A.

Timestamp order was chosen for message ordering to ensure that no federate received messages "in its past".

Logical Time Synchronized was chosen as the time-evolving method to allow a federate to participate in the time advancement of other federates and to allow other federates to participate in its own time advancement.

Event-Driven Federate time advance services were chosen to ensure events would be processed sequentially in TSO.

There are four different types of IRMs.

- **TYPE A: Entity Transfer** – This model and its sub-models represent interoperability problems related to entity transfer from one model to another in a distributed simulation environment. For ease of identifying the interoperability problem this IRM is sub-divided into Type A.1 (General Entity Transfer), Type A.2 (Bounded Receiving Element) and Type A.3 (Multiple Input Prioritisation). (The details of all IRMs are discussed in the next Chapter.)

- **TYPE B: Shared Resources** – This IRM represents the interoperability problems in sharing one or more resource(s) between two or more models in a distributed simulation environment.
- **TYPE C: Shared Event** – This IRM represents the interoperability problems related to sharing events between two or more models in a distributed simulation environment.
- **TYPE D: Shared Data Structure** – This IRM represents the interoperability problem related to sharing data elements or data structure between two or more models in a distributed simulation environment.

2.6.2 Summary of the related work

The CSP Interoperability Reference Model Standard presents templates and patterns that classify interoperability problems found when developing distributed simulations comprised of CSP models. The aim of the IRM standard is to establish a commonly understood basis from which CSP interoperability in distributed simulations can be evaluated and judged. IRMs simplify the assessment of present answers to CSP interoperability issues. They can help practitioners, model developers, and CSP vendors identify suitable solutions to such problems. IRMs also tackle several unaddressed interoperability issues identified in HLA. Taylor et al. (2002) underlined the basic problem of distributed simulation and stated, “If distributed simulation allows us to interoperate the two production model, then distributed simulation becomes a powerful technique that allows two (or more) companies to explore their relationships”.

Existing work in the area of addressing interoperability issues and modelling techniques, especially with discrete-event distributed simulation, is mainly fragmented, purpose-built, and often lacking the use of standards. Furthermore, while some solutions exist, they contain restrictions or are too specific, and they are also limited in reusability and composability.

In this research thesis, all interoperability issues identified in SISO-STD-006-2010 are addressed with different approaches using a standard HLA platform. The research also discusses the importance of interoperability issues during conceptual modelling. This research will not only help the modeller identify interoperability issues but will also propose a framework to address them. This section now discusses further contributions by different authors in addressing these IRMs.

Taylor et al. (2006a) proposed an approach to address IRM Type A.1 (General Entity Transfer) for COTS simulation packages. These simulation packages lack the ability to provide an entity transfer facility for distributed models, therefore a simple CSP Handler (CH) connected to the standard RTI was proposed to communicate between simulation models prepared with COTS packages. Taylor et al. (2006a) described a solution to IRM Type A.1 requiring minimum intercommunication, which involved transfer of timestamped entity messages from one model to another, the correct receipt of such entity messages from one or more models, and correct coordination with any events being processed by a receiving model's COTS simulation package.

Assumptions made in compiling the Entity Transfer Specification (ETS) for the basic Type A.1 model were: only one reception point in the destination model for a specific entity type received from a single specified source model; a CH (Figure 2.23) present in the Interoperability framework to provide a common data exchange format to convert to and from the ETS; time represented by the same units and resolution in both models; and entities defined by name and any attributes.

Wang et al (2006) proposed a solution to address IRM Type A.2, using a middle layer interface between each COTS model and the message-passing RTI called a DSManager. The purpose of the DSManager was to interpret and apply generic functions based on the HLA standard to be invoked by each CSP. The resolution provided by the PDG was to use DSManager as a status monitor enabling it to introduce small increments to simulation time when forwarding timestamped status messages. An additional variable time value was also added to the timestamp to denote any processing priority that receiving models might apply to two different entities. In the case of updating status, to allow for communication time and rapid status changes in complex environments, a small *lookahead* time was added by DSManager to the timestamp of the current simulation time. The software framework for operating such a distributed simulation federation was shown in Figure 2.23 describing IRM Type A.1. The difference is in the name of the middleware, i.e., DSManager, and the different function to the CH.

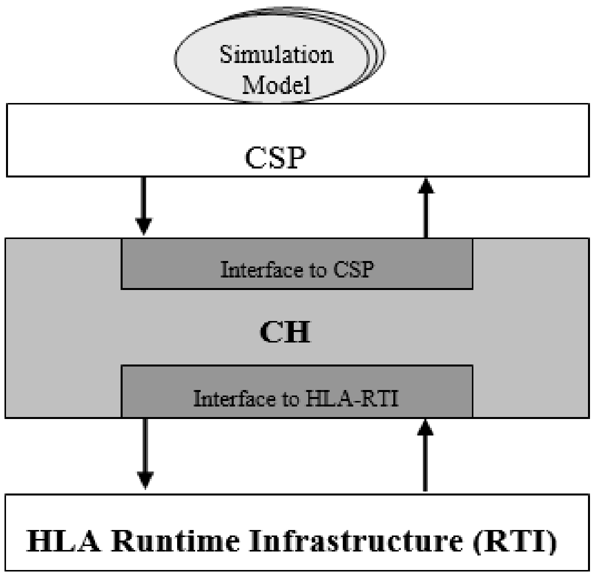


Figure 2.23: Reference Interoperability Framework (Taylor, 2006a)

Pedrielli et al. (2012) sought to modify and expand the protocols provided by the PDG for CSP interoperability as used in the civil sector. They focused on increasing functionality to meet unresolved situations by developing new solutions for entity passing and modifying the previous ETS. Pedrielli et al. (2012) based their proposed solutions on adopting or forming extensions to the previous work of the PDG, which involved a middleware adapter for communications between CSP and RTI and specific ETS protocols, as described earlier.

The concept of “shared state” or “shared resource” – IRM Type B – in distributed simulation processing is problematic due to the different simulated times at which federate models may be operating. When data variables that periodically change need to be accessed by participating federates, global issuing of updated data to all federates is inappropriate, so appropriately synchronized request and delivery mechanisms need to be devised.

Mehl and Hammes (1993) sought to introduce algorithms to give the illusion of consistent shared variables without the presence of shared memory. This approach involved the use of distributed shared memory (DSM) algorithms. Two DSM algorithms were suggested for use with conservative (no roll-back) distributed simulation: the *central server* algorithm, and the *read replication* algorithm. Neither of these algorithms used HLA standards implementation.

Turner, et al. (1998) proposed a resource conflict approach by creating conflict sets, assigning the resource weight, and then processing the simulation. This research assumed that the shared resources had multiple skills to operate multiple machines. The simulation was based on a deterministic approach in which resource utilisation was calculated before the simulation was run.

Low et al (2006) used a hybrid approach (i.e., a combination of decentralised and centralised) in which the owner of a shared variable allocated the resource and also kept a list (non-zero look ahead only) which was replicated to all other federates. Low et al. (2006) analysed four potential solutions that involved replacing the TSO request and reply messages with request order (RO) messages, plus the addition of a middle-layer manager for data file implementation.

Darma, Decentralised and Adaptive Resource Management, was the name of a proposal for optimising management of shared resource pools using mathematical algorithms, as presented by a research team at Dublin University (Loureiro et al., 2010). Although not specifically considering distributed simulation issues, this work addressed misapplication of resources in optimal terms. The authors generated solutions to the notion that general allocation of resources according to application demand was disturbed by varying application workload.

Some work has been presented on Shared Data – IRM Type D – but very little research has been conducted. Handling shared data in a distributed simulation environment is itself a big challenge. Some modellers confuse this IRM with Shared Resources or Type B. However, Type D is different and much more complex.

2.7 Summary

This chapter began with a description of the purposes and advantages of computer simulation. Different classifications of computer simulations were discussed and it was stated that a discrete, dynamic, and stochastic approach would be adopted in this research. DES terminologies were described and the advantages and disadvantages of the four further choices of approach available to DES were also explained. A history of DES software developments up to the present was accompanied by a brief listing of software products and suppliers.

The next section addressed M&S and noted that development of an appropriate and viable model was of equal importance as the ability to simulate its performance over time. Several illustrated examples were given of different analyses of the M&S project life cycle, from establishing real world problem and objectives to model conceptualisation and design, programming and testing, execution, and evaluation of results. The initial process of conceptual modelling was then examined in more detail with published examples of assistance to modellers of conceptual modelling frameworks and the principle artefacts involved.

Distributed simulation was then addressed with a description of its purpose and advantages and its specific requirement of simultaneous task execution by a group of separate simulators – meaning distributed simulation does not follow sequential time. The leading challenge for distributed simulation was presented as achieving interoperability between separately developed simulation models, with technical interoperability dependent on compatible hardware, standards, time management, and use of RTI. Reusability was identified as an important facility in encouraging wider use and development of distributed simulation. However, reuse was shown to be indicated for simple model features, such as code and basic sub-systems not complex complete models, with remedy dependent on common structures that enabled interoperability of models developed on different platforms. Composability, or the ability to model, create, and execute unique simulations from reusable system components, was also discussed with references to many theories, standards and frameworks that have arisen. Formalism in model and system design to address similar real world problems was illustrated with a table showing different types of modelling and the relevant development approach.

HLA was introduced, offering a common framework and standards for distributed simulation aimed at interoperability and reuse. HLA framework and rules were set out, as defined by IEEE 1516-2010 standard specifications, and the HLA concept of models as federates interacting in a federation, via a runtime infrastructure (RTI) API, was illustrated and described. The HLA OMT, FOM and SOM were also described together with the Run Time Infrastructure (RTI), which enables federates to publish and subscribe data with other federates.

The final section of this chapter presented the history of attempts by the Simulation Interoperability Standards Organisation (SISO) to create standards to assist in achieving interoperability between divergent COTS simulation packages. The current major effort from the CSPI PDG was then presented by focusing on the group’s development of IRMs, which identify the generic specifications of particular interoperability problem types including several subcategories. The four major IRM types were described and an example given of how these IRMs might be applied in the case of military distributed simulations. This was followed by a summary of other research attempts at resolving the problem types identified by the PDG IRMs.

The next chapter discusses data and communication in simulation modelling. It also highlights available modelling techniques by offering some comparisons and their suitability for use in distributed simulations.

Chapter 3: Design of Interoperability issues to address IRMs

3.1 Overview

This chapter addresses the process of designing a working DSI framework for the use in distributed simulation industry, that incorporates the interoperability problem-solving methods described by IRMs.

The chapter begins by discussing the problems of data exchange, communication, representation, and time synchronisation when conceptualising and developing a distributed simulation model.

The chapter proceeds with an overview of the need for an DSI framework due to the common practice of COTS simulation packages using vendor-specific libraries, components, and reference models, and different conceptual and engineering approaches. Specific interoperability issues are discussed and optional approaches to simulation configuration outlined.

The research technique adopted in this thesis is then established as following recommended IEEE standards including a modification of the DSEEP.

This simulation design approach is then divided into each stage of the DSEEP developmental process with an explanation of each procedure.

A discussion on conceptual modelling and interoperability is then followed by a concluding section explaining the objectives of each of the six IRMs.

3.2 Issues in Modelling Interoperability

Interoperability refers to the facility to exchange data or information between two or more systems or models. Interoperability is often confused with integration, because both impact on enabling seamless and collaborative information flow between systems. Integration is connecting different models together with a communication infrastructure that enables the models to interoperate. The interoperability focus is on maintaining continuous communication between the systems. Therefore, to achieve integration requires successful interoperability, i.e., both the interface and specifications of the two systems must match. For example, the power plug in Figure 3.1 (on the left) is for a mobile charger. To charge a phone the power plug needs to be integrated with the wall socket (on the right). This is not possible because they are not compatible. So to interoperate, an adapter is required (centre) that helps connect the power plug to the wall socket.

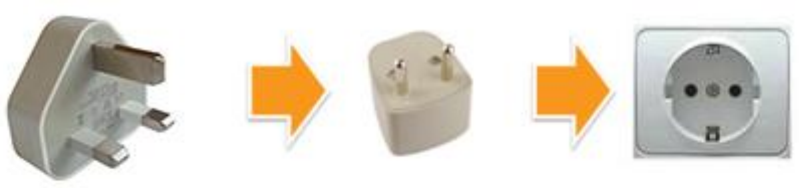


Figure 3.1: Interoperability Challenge

Hence, interoperability is required to achieve a degree of compatibility between modelled systems. Systems using standard communication techniques, such as HLA, should ideally be automatically compliant with all major interoperability issues such as data exchange and interpretation of transferred data. However, this is not truly possible in practice, because of rapid changes in technologies, legacy systems, hardware changes, new programming languages, etc. The simulation research community has accepted this fact and has proposed system abstraction to hide interoperability and implementation details. This section further describes two special conditions required to provide interoperability between systems.

3.2.1 Data exchange, communication and representation

Achieving data integration between the sub-systems in distributed simulation is a challenge. It is also the case that distributed simulation is comprised of sub-models or sub-systems representing individual simulations themselves. Thus, it is important that all participants must agree on a common interpretation and data exchange. At the abstract level this has been addressed with HLA object models. The interaction and attribute classes are defined by the FOM. These classes define how the data will be interpreted, e.g. 64-bit integer, 6-byte string, etc. The FOM can standardise the data objects and variables at interaction level, but it is not necessary that these objects or variables are similar to the internal data type used in each model (e.g., the size of a character in C++ is 1 byte while in Java it is 2 bytes).

Another challenge concerns the ownership of the data element, i.e., who is allowed to change the value of a shared data element. HLA has provided a partial solution to this problem by requiring object / variable ownership to follow publish and subscribe methods before a simulation starts, i.e., object ownership and control need first to be identified by each participating model comprising the simulation federation. HLA also enables dynamic ownership, i.e., ownership can be transferred during the simulation run. However, changing values due to ownership of an object / variable by more than one sub-model or federate could lead to conflict or deadlock. The implications of multiple ownership can result in problems identified in IRM Type B (shared resources) and Type D (shared data structure). Further, conflicts can be anticipated on how such data is represented (syntactics) by an individual model and how a model interprets the meaning of that data (semantics). This section further discusses these abstractions, representation, and interpretation concerns.

3.2.1.1 Data Values

Data value conflict is a common problem faced by simulation modellers. The causes relate to the use of different data types of different programming interfaces and different forms of representation affecting how data is interpreted (e.g., type mismatch, allowed values, abbreviations, etc.) There is no single “fit for all” solution to this conflict because of constantly emerging technologies. Therefore, clarity of understanding must be established when using these data values with FOM objects, and if necessary conversion methods should be introduced.

3.2.1.2 Data Models

Data model conflicts arise due to different programming approaches to modelling data. For example, one model is designed using structural language while another is developed using object-oriented language. Similarly, with hybrid simulations, one model may be developed using Agent-Based Simulation (ABS) while the other is developed using DES practice. Both the models should allow for the complete semantics (data representation), otherwise information could be misinterpreted or lost. However, one way to address this conflict is to focus on the definition of the data model at an abstract level.

3.2.1.3 Schema

Schema conflicts can occur due to misunderstanding of the data object. For example, the definition of a vehicle in one model is based on properties or attributes related to objects that drive on the road, while another model may also include properties or attributes of other vehicle types, including planes and boats. Hence, understanding of the speed unit could involve a mismatch between car and boat, since car speed is measured in kilometres or miles per hour while boat speed is measured in knots. Another conflict is the use of different names for the same element, such as the use of “vehicle” or “car” for the same real world object. There can also be a situation where a metadata from one schema can be a data in the other schema. Although this problem has been addressed by the HLA FOM at a more abstract level, the problem remains at the model syntactic level.

3.2.1.4 Semantic and Syntactic

Semantic issues are defined as the way models interpret the meaning of data exchanges between models. Syntactic issues are defined as how that data is represented. Syntactic conflicts can arise at the data model level or the schema level. Semantic conflicts may occur at any level of abstraction, i.e., schema, instance, or data value. Therefore, both semantic and syntactic conflicts can exist in some situations, but they should be tackled independently. One approach is to maintain interoperability of models at the most abstract level possible to avoid such conflicts.

3.2.2 Time Synchronisation

One of the major issues in distributed simulation is the synchronisation of time between the participating models. HLA therefore provides a time management service in its interface specification. It is easy to manage time in a standalone simulation system because the simulation progresses sequentially, but in distributed simulation the process progresses in parallel. HLA establishes synchronisation by providing an advance time facility within its time management service. The reason is that, in reality, even similar hardware (e.g., PC) may not have the same performance and speed. The process routines of models might also differ. Each participating node (PC) will try to run its model as fast as possible, so it is possible that one model needs to wait for another model to reach its required status. This is managed by the time advance feature. Although HLA achieves time synchronisation, it is still the individual model that decides what data or information needs to be exchanged at a particular time. Data exchange becomes more complicated when the simulation requires more than just simple message passing as indicated in the IRMs. A further consideration affecting time synchronisation is the approach used by the modeller. In distributed simulation, there are two traditional time synchronisation approaches used, apart from the Hybrid approach, these are Time-stepped synchronisation and Real-time synchronisation.

3.2.2.1 Conservative Synchronisation

Conservative Synchronisation was the first synchronisation technique designed to judge when it is “safe” to process an event. In this approach, time does not advance as real clock time, but is advanced when an event needs to be processed in non-decreasing time stamp order to avoid local causality. The Conservative approach also uses a lookahead variable to specify how far ahead the system will schedule an event. A zero lookahead means the event with the smallest timestamp can be processed. The first synchronisation algorithm Chandy-Misra-Bryant (CMB) was proposed by Bryant (1977) and Chandy and Misra (1979). This paper's research is conducted using the Conservative synchronisation approach and most of the implementation is with zero lookahead.

3.2.2.2 Optimistic Synchronisation

In the Optimistic synchronisation approach, event causality is initially allowed, but is then reversed, i.e., this approach allows "unsafe" events to be run, which are later detected and recovery made from any causality error. The optimistic synchronisation approach exploits the greater degree of parallelism, but with the cost of potentially creating more computational overhead, which can also seriously affect performance. Both conservative and optimistic synchronisation is supported by HLA.

3.2.2.3 Event Synchronisation

Event synchronisation is an important factor to consider when creating a simulation using the discrete event approach. Events are instances in time, therefore once an event is scheduled the simulation clock is advanced to the next scheduled event. There is a possibility that more than one event is scheduled for the same time. In such a situation the simulation must decide which event should be scheduled first to maintain the proper sequence of events. This is because one event could affect another, for example, it could cancel the other event. Event management is outside the scope of HLA, and event clashes could seriously affect interoperability between participating models.

3.3 Requirements for Interoperability

Before establishing the importance of interoperability standards in distributed simulation, we need to establish the need for distributed simulation and consequent interoperability standards when specialised and optimised tools for developing simulations are already in the market.

Many software vendors provide simulation packages with predefined libraries specific to their package. These libraries might consist of common built-in components which can be used in different simulation scenarios, e.g., a conveyor system. Such libraries are either available with the packages or as add-ons. To communicate between these sets of libraries the vendors establish their own reference models. These reference models provide a specific standard, which only operates with their package. Thus, with each vendor, creating their own reference model, their simulation software will not be compatible with other vendor packages. Hence, to assist re-usability and scalability we need a standard reference model for interoperability. Another reason for using distributed simulation is the possibility for a business to hide its processes from competitors or the outside world, e.g., in a supply chain of multiple businesses. Having standard IRMs can help establish the process-hiding facility for the business.

In Chapter 2, the theory of distributed simulation was discussed and the most advanced approaches for distributed simulation were presented. It was established that for interoperability and re-usability a standard platform is required, and that HLA defined by IEEE 1516-2010 is the most appropriate known standard for integrating simulation models.

As stated previously, the design and development of HLA originated with the US defence community. The DMSO have been responsible for integrating many types of potential defence-oriented simulation models (Kuhl et al., 1999). In 1996, the DoD chose HLA to be the standard for their simulations (DMSO, 1998a, 1998b, 1998c). HLA was later accepted in 2000 by IEEE as the distributed simulation standard IEEE 1516 (IEEE, 2000a, 2000b, 2000c).

Due to the continuous evolution of distributed computing and to meet the demand for increasing complexity in large scale simulation systems, the HLA standard widened its scope to handle the interoperability of such simulation models. Large organisations, such as the DoD Modelling and Simulation Coordination Office (DMSCO), and the Simulation Interoperability Standards Organisation (SISO), are continuously involved in expanding the potential of distributed simulation by identifying challenges faced by industry.

SISO arranges extensive forums to educate the M&S industry about simulation interoperability. SISO also helps the development of standards and arranges large workshops and conferences on simulation interoperability, such as the annual European Simulation Interoperability Workshop (SIW) and the semi-annual Simulation Interoperability Workshop. Despite SIW focusing on simulation interoperability only, these workshops have drawn large numbers of attendees, comparing favourably with the WSC, which is the largest annual simulation event with a much broader simulation scope. Additionally, under the umbrella of SISO, many product development groups and study groups have begun with an objective of investigating simulation interoperability issues from different viewpoints.

This activity confirms a considerable effort toward developing the interoperability of simulation models and the use of HLA as a common application. However, the use of HLA remains generally within the defence domain and there is still little awareness in other industries. The reason for this fact is not clear given HLA promises to provide such facilities as reuse of existing components, interoperability, provision for information hiding, and integration of heterogeneous models. Although the defence community initiated the design and development of the HLA standard, this significant effort was also planned to support the industrial community. It has been noted that HLA is rarely applied in the industrial sector (Bore et al., 2006), therefore, several panel discussions have been held at the WSC to investigate the reasons behind this phenomenon (Taylor, et al., 2002, 2003a). In addition, a forum named HLA-CSPIF10 was created to study the usage of distributed simulation in industry.

It is assumed that COTS simulation packages are used for most commercial simulation projects. This premise is based on different researches conducted within the industry and on discussions with simulation practitioners at several conferences and workshops. The general reflection is that COTS simulation products rarely support HLA, and that vendors rarely address distributed simulation (Bore et al., 2008). This may be why distributed simulation or HLA cannot be found in the industrial market, i.e., because most simulation practitioners (understood to normally use COTS packages) do not have the option or knowledge to implement it.

Furthermore, there also appears to be lack of demand from simulation practitioners for such distributed simulation services as transparent HLA user interfaces. Perhaps this is because absence of the proper tool means its benefits cannot be appreciated. This may suggest that lack of engagement by industry with distributed simulation is a supply and demand problem. But this is another assumption.

Distributed simulation does not only face the issues of interoperability and reusability, but also the influence of COTS simulation package vendors. COTS simulation package vendors identify few successful distributed simulation projects and consider that distributed simulation broadly is rarely used in industry (Boer, et al., 2008). In a survey conducted by Csaba Attila Boer (2005), approximately only half of the COTS vendors or their customers claimed success with distributed simulation projects. Undoubtedly, COTS simulation package vendors adopting the distributed simulation concept would involve cost to either purchase the interoperability tools or develop them and additional time to design and develop the distributed models. At present, most COTS simulation package vendors do not support distributed simulation features and complain of increased costs to incorporate the tool (Bore et al., 2008), although a counter argument points out that DMSO RTI has been available as a free tool since 1996.

Monolithic models are much easier to build than distributed models. Distributed models involve the additional understanding of time synchronisation, data representation, data exchange, management of object ownership, etc. In monolithic simulation models most of these factors would not apply so a regular modeller would be unfamiliar with these complex concepts. Furthermore, to standardise the model for distributed simulation the modeller needs to understand and become familiar with the concepts of HLA. This all adds additional time requirements for design and development.

HLA was introduced by the DoD to provide an architecture for simulation interoperability and the reuse of defence projects. But practical use by wider industry requires more work and time on the semantic inconsistencies that can occur during inter-operation between models. Many experts have identified the unsolved interoperability problems, where the main obstacle to practitioners using distributed simulation is aligning different data models. According to one expert, "Semantic interoperability is a big issue. This is a hard problem, and probably much harder than the problem for which HLA was designed and developed" (Bore et al., 2008). Indeed, distributed simulation interoperability is a known issue for the system engineering community (Boer and Verbraeck, 2003).

A survey was conducted by experts and industry specialists (Boer et al., 2008) to identify the issues faced by COTS simulation package vendors. The majority of responses from industry experts identified the complexity of HLA. This research also concluded that the previously identified issues of cost, i.e., acquiring and embedding HLA (RTI) in COTS packages, was not an issue. Instead, both vendors and industry believed that the HLA interface was too complex and technical, and it resided at too low ~~a~~ level in the system hierarchy. They believed it would not be possible for a regular modeller to use vendor packages to develop distributed simulation models. As identified previously, distributed modellers would still need to take other interoperability issues into account. A defence expert claims that "there is no incentive for different vendors of different simulation packages to agree on a common interoperability standard" (Bore et al., 2008), but if HLA functionality became more abstract, it would lower the learning curve required by modellers and boost the potential of HLA within the industry.

In 2014, NASA launched the Simulation Exploration Experience (SEE) project in collaboration with scientific organisations such as SISO, Liophant, SCS, and Simulation Team, with the support and participation of many professional associations, industry bodies, universities, and students. SEE is a distributed simulation challenge in which a number of dispersed inter-university teams with their technical partners work in collaboration to design, develop, test, and execute a simulated lunar mission. The sole aim of this challenge is to the importance of distributed simulation interoperability, especially in large scale projects. The SEE evolved from the Smackdown project, initiated in 2011, which focuses on COTS simulation packages like Pitch and VT MAK. Teams composed of company internships and university students and departments develop different simulators, based on HLA Evolved, to be integrated into the Smackdown Federation (Taylor et al., 2014) (Garro et al., 2015).

As indicated by the discussion above, in fact there have been various attempts by COTS simulation package (CSP) vendors to create a distributed simulation environment based on HLA standard. Strassburger et al. (1998) was first to address the problems faced by CSPs using HLA to create distributed simulation. The standard is complex and it is difficult to assess how models may interoperate (Taylor et al., 2006). Therefore, the SISO CSPI PDG developed and standardised a set of IRMs. The objective of these IRMs is to reveal the interoperability issues faced by the modeller when designing and developing a distributed simulation model. (Details of IRM standards are listed in the previous chapter.)

CSPI PDG has defined the IRMs as creating a common frame of reference "to help vendors and industrial experts to achieve solutions to complex interoperability problems". There have been many different and isolated approaches proposed to address the issues identified in IRM SISO-STD-0006-2010 standard. The possible solutions, different methods and implementations are extremely difficult to capture. The next section will discuss these approaches in more detail to emphasise the need for the proposed framework.

3.4 Research Technique

It has been established that M&S is the backbone of operational research to address the challenges of a highly complex and dynamic simulation environment (Tolk et al., 2007). Chapter 2 described various approaches for planning and building a distributed simulation environment. M&S is used by a wide spectrum of users and systems to support personnel training, analyse logistic issues, test and evaluate new systems, and for many other objectives. In general, M&S is not only used as an exercise to document a process, but also to reduce system development risk, and to save cost. Documentation is necessary to understand, modify, and reuse the system or system components, while risk is calculated during the conceptual phase when the stakeholder makes a decision on whether the system is required. Thus, the purpose of this risk assessment exercise is to make a wise decision on how and what system should be designed, developed, and implemented.

There are several factors that influence the choice of system engineering methodology that works best in developing a distributed simulation. These include security issues, class of application (e.g. experimental, training, testing, etc.), development team experience, cost, M&S assets (verification, validation and acceptance), etc. While the distributed simulation community continues further research in the sixth era of integration, using different methodologies to support different M&S assets, the community still has not reached its goals. One way to further enhance the process is to harmonise commonalities found in different distributed simulation methodologies and bring these together with additional areas of research. This is the approach used in this thesis.

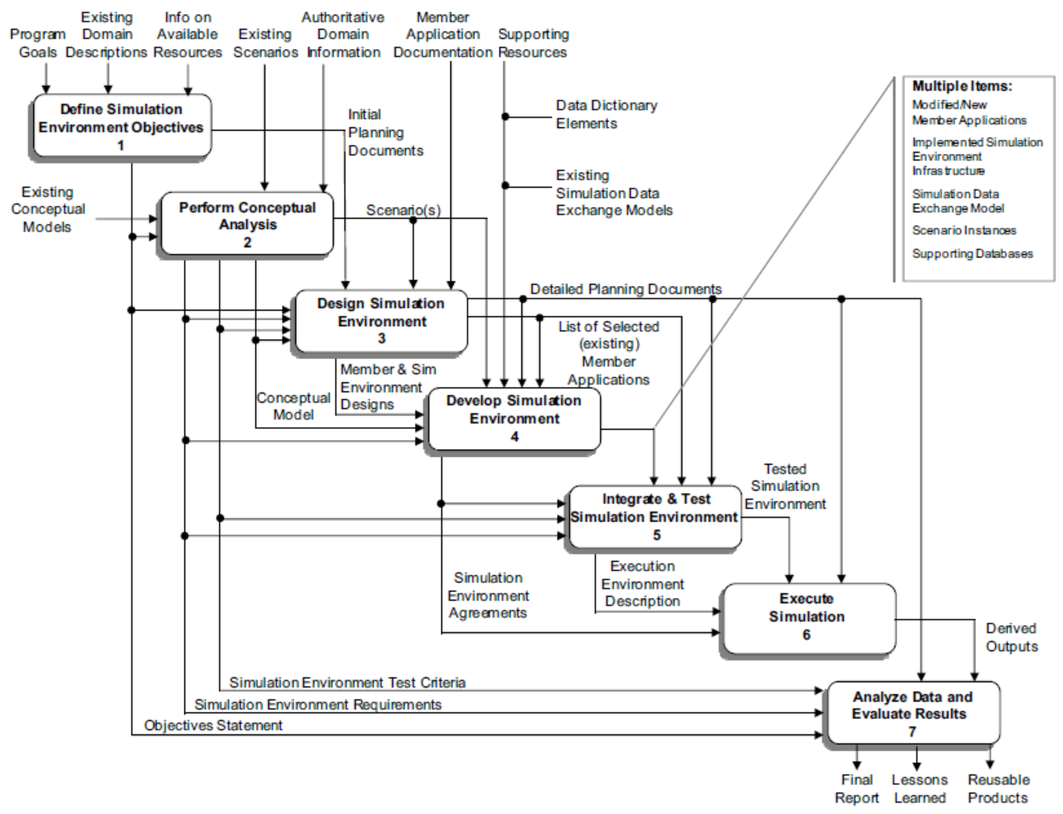

Program Goals
Existing Domain Descriptions
Info on Available Resources
Existing Scenarios
Authoritative Domain Information
Member Application Documentation
Supporting Resources
Define Simulation Environment Objectives 1
Initial Planning Documents
Data Dictionary Elements
Existing Simulation Data Exchange Models
Existing Conceptual Models
Perform Conceptual Analysis 2
Scenario(s)
Multiple Items:
Modified/New Member Applications
Implemented Simulation Environment Infrastructure
Simulation Data Exchange Model
Scenario Instances
Supporting Databases
Detailed Planning Documents
Design Simulation Environment 3
List of Selected (existing) Member Applications
Member & Sim Environment Designs
Conceptual Model
Develop Simulation Environment 4
Tested Simulation Environment
Integrate & Test Simulation Environment 5
Execution Environment Description
Simulation Environment Agreements
Execute Simulation 6
Derived Outputs
Simulation Environment Test Criteria
Simulation Environment Requirements
Objectives Statement
Analyze Data and Evaluate Results 7
Final Report
Lessons Learned
Reusable Products

Figure 3.2: Distributed Simulation Engineering and Execution Process (DSEEP) (IEEE Std., 2010)

The majority of this research follows the simulation methodology defined by IEEE Std. 1730-2010 and IEEE Std. 1730.1-2013 (IEEE Std., 2010, 2013). The recommended standard practice for DSEEP is illustrated in Figure 3.2. The IEEE recommended approach is based on research from different M&S techniques proposed by different authors, but is more specific to distributed simulation and that is why it is used for this research. In the majority of modelling technique, literature is related to non-distributed simulation environments. This chapter will identify how this approach is used, with slight modification, to address distributed simulations. In identifying the importance of interoperability in distributed simulations, it was also discovered that Validation and Verification was an item missed out in Stage 5 (i.e., Integrate & Test Simulation Environment). This thesis will discuss a framework proposed by Robinson (2014) to understand how the proposed conceptual modelling stage could be further refined to suit the needs of distributed simulation.

3.5 Simulation Approach

The approach used in this research (based on IEEE Std. 1730-2010) consists of 7 steps, which are illustrated in Figure 3.2. These steps were briefly discussed in the previous chapter with an abstract view of the approach shown in Figure 2.9. This section will detail more thoroughly how these steps were used to develop the proposed framework and the approach adopted to manage interoperability issues within a distributed simulation environment. There are four main types of identified interoperability problems (Types A, B, C, and D), but Type A is further divided into three sub types. Therefore, in total, there are six IRMs. A solution to each IRM was developed using the simulation techniques described below. Each IRM was addressed in isolation for simplicity and to better understand the process of modelling through different stages of the life cycle. In a real world system, there can be more than one IRM applicable. As an example, an Emergency Medical System that was developed by Anagnostou (2014) using hybrid distributed simulation models. The two major models in the system represented an A&E hospital department and an ambulance service. The ambulance service was responsible for collecting patients and delivering them to A&E. Therefore, Type C IRM (shared event) was used in the ambulance model to announce all vehicles available to collect a patient. The A&E model used both Type A.1 (General Entity Transfer) and Type A.2 (Bounded Receiving Element) to manage the transfer and the reception of each patient delivered by ambulance to the hospital.

3.5.1 Define simulation environment objectives

Like other simulation development approaches, the focus of this stage is to understand the problem domain. A need statement is prepared which includes key events and environment conditions, initial estimates of required reliability, high level description of critical systems, and input and output data. Most importantly, it should list the available resources (e.g., funding, personnel, facilities, etc.), and known limitations (e.g., deadline, holidays, restriction to data access, etc.) to underlining the simulation environment.

A good approach to model a system is to keep it simple (Pidd, 2004), because in becoming more and more complex it also becomes less understandable. Hence, in this research the problems identified by IRMs (i.e., issues related to interoperability between distributed simulations) are addressed individually. Each of the six IRMs, as described above, will be modelled separately providing six different Problem statements / objectives. Each of these problems will be addressed by following a complete iterative process of the selected development life cycle.

1) Problem Statement for Type A.1: General Entity Transfer

2) Problem Statement for Type A.2: Bounded Receiving Element

3) Problem Statement for Type A.3: Multiple Input Prioritization

4) Problem Statement for Type B.1: General Shared Resources

5) Problem Statement for Type C.1: General Shared Event

6) Problem Statement for Type D.1: Shared Data Structure

3.5.2 Perform conceptual analysis

This stage develops the non-software-based representation of the real world model. During this stage the modeller will develop a functional specification of the scenario. According to the simulation approach (e.g., discrete event, agent-based, system dynamic), multiple scenarios are defined as part of the functional specification.

IEEE Std. 1730-2010 recommends introducing the Simulation Data Exchange Model (SDEM) at Stage 4 during the development of the simulation environment. However, this research proposes and implements using SDEM going forward from Stage 2 of the development life cycle to find the interoperability problems defined by IRMs, and the levels of interoperability required (Talk, 2003), as functional specifications arise from the multiple scenarios identified during this stage. The SISO's Conceptual Modelling Group believes that interoperability should be identified at conceptual modelling level (Borah, 2006). Similarly, Wang et al. (2009) also emphasise identifying levels of interoperability (Talk, 2003) at conceptual modelling stage.

A further reason for drawing up the interoperability study at this stage is the link between Conceptual Analysis (Stage 2) and Integration and Testing of Simulation Environment (Stage 5). (More details about this relationship are given in section 3.5.4 below.) Establishing the need to identify interoperability at conceptual modelling stage leads us to a further issue, namely, when and how to identify interoperability issues during conceptual modelling. This is further explained in section 3.5.9 (Conceptual Modelling and Interoperability) where a conceptual modelling approach identified by Robinson (2014) is modified to suit the new requirements. Meanwhile, the IRM problem list represents the interoperability issues for which requirements and conditions will be prepared individually for each problem statement.

3.5.3 Design simulation environment

The main objective of Stage 3 is to produce the design of the simulation. During this step, the suitability of individual simulation systems is determined. These individual simulation systems represent the multiple scenarios identified in the conceptual modelling step. Potential entities and events for each member application (termed model in this research) are identified. Reusable member applications can be discovered and arrangements for integration put in place. The final selection of member applications is influenced by two constraints: Technical (e.g., V&V) and Managerial (e.g., security, facility, and availability). Further, only once the individual simulation has been identified (in the previous step) can the approach be determined to address interoperability between the member applications (models in the IRMs).

3.5.4 Develop simulation environment

This step has three main objectives:

- Develop simulation data exchange model (SDEM);
- Establish simulation environment agreements; and
- Implement member application designs.

As discussed above, the author proposed introducing SDEM during the conceptual modelling stage to identify interoperability issues and needs, whereas the IEEE standard recommends introducing the SDEM at Stage 4 of the development life cycle.

The recommendation states that SDEM identify the runtime data exchange between the member applications and an agreement on how these members will interact with each other. In this research, identifying and selecting such agreements was completed in earlier steps. Therefore, focus stresses development and implementation – based on the identified interoperability issues.

3.5.5 Test Simulation Environment

The purpose of this stage in the DSEEP approach is to:

- Fully describe and plan the execution environment;
- Test the interoperability between all member applications; and
- Test the simulation environment for achieving the core objectives.

A complete set of information and agreements defined in SDEM, and detailed functional specifications and interoperability requirements from conceptual modelling, are tested, verified, and validated.

To achieve this, three levels of testing are defined, which contribute toward the overall system of V&V.

1) Member application testing: During this testing each member application is tested for expected performance as defined earlier in the conceptual modelling.

2) Integration testing: During this testing, the integrated simulation environment is tested to verify the level of interoperability.

3) Interoperability testing: During this testing, interaction between the member applications is tested to ensure it accords with the defined scenario.

Testing is achieved by evaluating system compliance with the specified requirements. In other words, testing is only possible if there is a set of defined criteria with which to compare the results. Therefore, the statement in Step 2 recommending linking these two steps together is now revisited. Effective interoperability testing is only possible at this stage if the detailed interoperability requirement specifications have been identified earlier, i.e., during the conceptual modelling stage, similar to the remaining two testing levels.

3.5.6 Execute simulation

After necessary amendments as required following the testing step, the simulation must be executed as a set of member applications (i.e., as one simulation). Key simulation test criteria should be executed to evaluate the success of the execution. The successful execution result in data output is collected and pre-processed (i.e., data formatted) for the next stage where the output will be analysed. Further execution might be required if erroneous data is suspected from any execution run.

3.5.7 Analyse data and evaluate results

This step involves the evaluation of data collected during the execute simulation step. To analyse the data collected, appropriate methods or third party, COTS tools can be used. It is a possibility that the required analysis cannot be achieved by any single available tool, therefore either specialised analysis tools could be developed or a combination of tools could be used. The tools required for data analysis must be identified during the conceptual modelling step with the resource requirement. Similarly, the analysed data must be evaluated against the objectives set in the conceptual modelling step. This analysis and evaluation step determines if the simulation is "passed / failed".

3.5.8 Verification and Validation process

Verification and Validation (V&V) is one of the most important processes. The system analysis or M&S community was the first to apply technical discipline to struggle with the terminology of V&V (Oberkampf and Roy, 2010).

Verification

The US DoD defined Verification as "*a process of determining that a model implementation accurately represents the developer's conceptual description and specification*" (Engel, 2010).

Validation

Similarly, the DoD definition of Validation is as "*a process of determining the degree to which a model is an accurate representation of the real world from the perspective of intended uses of the model.*" (Engel, 2010).

In software engineering, verification triggers at the evaluation phase to examine if the product meets the requirement specifications. Petty (2009) defined verification in M&S as "*the process of determining if a model is consistent with its specification*". In other words, it establishes the correctness of the model. This definition does not completely satisfy distributed simulation modelling. As there are multiple applications (models), each representing a simulation, the verification is not limited to the individual model correctness, but also deals with the correctness of interoperability between the models. Hence, the proposal in this research for earlier identification and selection of interoperability issues in the development life cycle to determine models by how they interoperate with each other against the given specification.

Validating a model deals with building the correct model, i.e., accurately representing the real world system. This is achieved by model testing. Testing will ensure the system is an actual representation of the real world. Hence, Stage 5, as explained above, has three levels of testing, including interoperability, to validate the model as a whole.

3.5.9 Conceptual Modelling and Interoperability

Communication between models can drastically affect performance and outcome, thus to avoid intercommunication problems, analysts need to improve the ability of models to share or exchange information, i.e., improve interoperability (Leal, 2017). Leal (2017) further proposed using a holistic interoperability assessment approach based on the interoperability dependencies for any given simulation. SISO-STD-006-2010 lists templates or patterns as IRMs that classify specific interoperability problems for such environments. There are two clear objectives for this standard:

- *"to clearly identify the model/CSP interoperability capabilities of an existing distributed simulation" (*SISO-STD-006-2010).
- *"to clearly specify the model/CSP interoperability requirements of a proposed distributed simulation"(* SISO-STD-006-2010).

This standard presents common standardised templates or patterns to identify interoperability problems in distributed simulation as selected by a team of industrial specialists. Hence, we have a list of known interoperability problems faced by the distributed simulation modeller community.

If we summarise the discussion, the modelling of interoperability for distributed simulation can be divided into the following four phases, ~~also~~ illustrated in Figure 3.3.

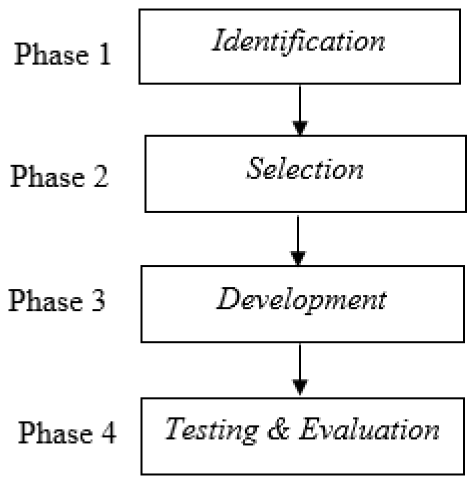


Figure 3.3: Interoperability Phases

1. *Identification:* to identify which IRMs are required for the proposed simulation environment. This phase must be part of Stage 2 (Conceptual Analysis) of the development life cycle.
2. *Selection*: to select the approach to address the IRMs. This phase must be part of Stage 3 (Design Simulation Environment) of the development life cycle.
3. *Development*: to apply the selected approach. This phase must be part of Stage 4 (Develop Simulation Environment) of the development life cycle. This phase requires outlining the exact data structure (i.e., SOM, FOM, and component level variables) and the level of access for each model to interact with each other based on the approach selected in the previous phase.
4. *Testing / Evaluation:* to verify, validate, and evaluate the proper interoperability between the member models. This phase must be part of Stage 5 (Testing) and Stage 6 (Evaluation) of the development life cycle.

3.5.9.1 Levels of Conceptual Interoperability

"Conceptual Interoperability between two machines is hard to achieve, but this must not stop us from trying" (Wang et al., 2009).

We can at least identify how close the system is to achieving conceptual modelling. Therefore, Tolk and Muguira (2003) first proposed *Level of Conceptual Interoperability Model* (LCIM), as a framework towards conceptual interoperability and composability. The seven levels of conceptual interoperability are illustrated in Figure 3.4. This figure is the latest version of the LCIM framework presented by Wang et al. (2009). From the bottom upward, this figure illustrates the increasing capability for interoperation, from Level 0 representing No Interoperability to Level 6 achieving Conceptual Interoperability. The LCIM framework suggested that HLA standard can be used to achieve up to Syntactic Level 2, while the BOM (Base Object Model Standards – SISO-STD-003-2006) is used to improve the composability, reusability, and interoperability at component level only. Hence BOM can exist at Dynamic Level 5 (Wang et al., 2009). This research is also used by different authors, most recently Kostelic (2017), to implement a digital Library Ecosystem. The LCIM framework also serves two descriptive and prescriptive functions defined by software engineering.

The descriptive roles document the interoperability and levels of interoperability between the models, while the prescriptive roles describe the requirements and approaches to achieve conceptual representation between models. Hence, the descriptive role can be applied at the Conceptual modelling phase to identify and document interoperability, and the outcome can be used to evaluate interoperability in the later stages of the development life cycle. Page et al. (2004) categorised the same model in three distinct layers.

- *Integratability:* This layer deals with the physical connection.
- *Interoperability:* This layer describes implementation issues, e.g., data exchange.
- *Composability*: This layer operates at the component / model level.

We already have established that to achieve composability in distributed simulation the systems must be interoperable, but composability is not necessary for a system to be interoperable (Page et al., 2004). All models are not necessarily composable but they still need the interoperability to function in a distributed environment.

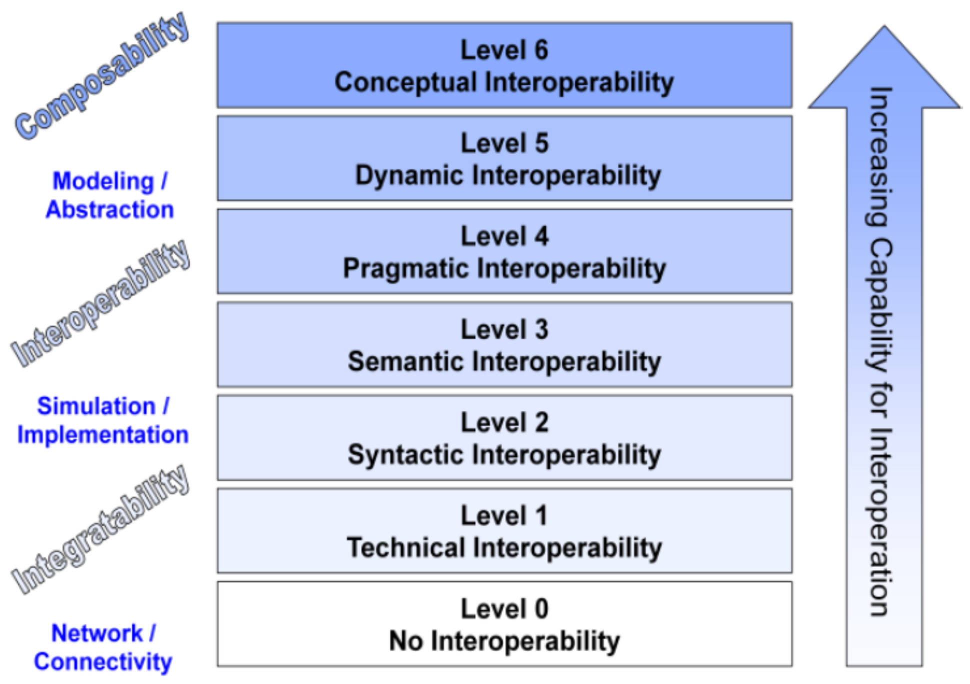

Composability
Modeling / Abstraction
Interoperability
Simulation / Implementation
Integratability
Network / Connectivity
Level 6 Conceptual Interoperability
Level 5 Dynamic Interoperability
Level 4 Pragmatic Interoperability
Level 3 Semantic Interoperability
Level 2 Syntactic Interoperability
Level 1 Technical Interoperability
Level 0 No Interoperability
Increasing Capability for Interoperation

Figure 3.4: The level of Conceptual Interoperability Model (Wang et al., 2009)

3.5.9.2 Conceptual Model for Distributed Simulation

Robinson (2015) stated that "it is useful to identify the requirements for generic conceptual frameworks". Developing distributed large scale and complex simulation models was not in the scope of work presented by Robinson (2008, 2015). Nevertheless, it is relevant to consider the work in Figure 3.5 below, showing the relationship between model *complexity* and model *accuracy* by Robinson (2008a). The research in this thesis focuses on model accuracy only, but for further discussion on model complexity the reader's attention is drawn to Pidd (2010).

Reaching the optimum level of accuracy of a model is marked by point x in Figure 3.5. Going beyond x will add further complexity, but might begin to lose accuracy. Another way to reach the model objectives is to distribute the model into smaller and less complex models, where individual models can reach up to maximum accuracy vs. complexity, in other words *point x*. Using smaller models is also advantageous for further development by reusing them in another complex model. Distributing the model requires addressing the interoperability challenges, as identified in the previous chapter. It implies that the type of interoperability required for a distributed model needs to be identified before any further analysis. This could be established during the Conceptual modelling stage.

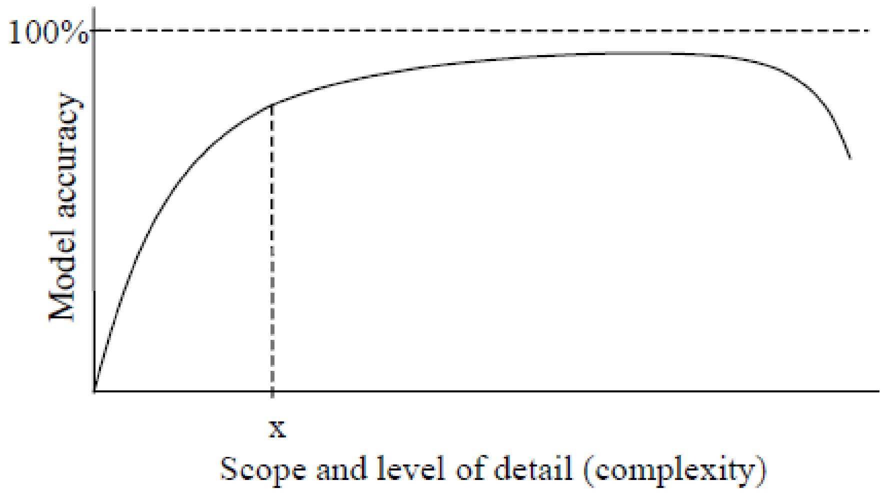


Figure 3.5: How simulation model accuracy changes with the complexity of the model (Robinson 2008a).

Capturing all the elements required of conceptual models is not easy. Robinson stated that there is "no right conceptual model for any specific problem" (Wang et al., 2009). Robinson, (2008a) also described conceptual modelling as a non-software specific descriptor of computer simulation. This specific description of computer simulation includes objectives, inputs, outputs, content, assumptions, and simplification. But for distributed simulation during the conceptual modelling stage *interoperability* should also be included in the specific description. Hence the definition of conceptual modelling in distributed simulation could be reworded as:

> "A non-software specific description of the computer simulation model (that will be, is, or has been developed), describing the objectives, inputs, outputs, content, assumptions, ***interoperability***, and simplification of the model."

Earlier in Chapter 2, conceptual modelling was explained in detail, but how conceptual modelling is used to develop distributed simulation was unspecified. By now, we already have established that interoperability must be identified at the conceptual level phase, but we are left with questions of how and when to identify interoperability within the conceptual modelling phase. Figure 3.6 answers these questions.

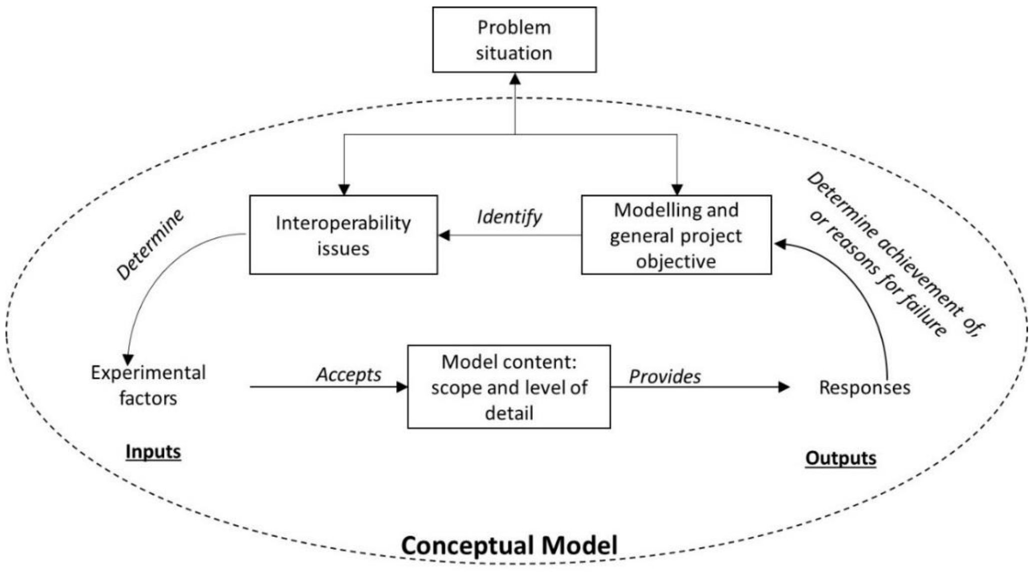


Figure 3.6: Robinson's Conceptual Model Framework revisited

The Robinson (2014) conceptual modelling framework (see Figure. 2.12, Chapter 2) is revisited to illustrate an extra activity of interoperability. The original conceptual model consisted of four main components, but the proposed approach comprises five main components: objectives, interoperability, inputs (experimental factors), outputs (responses) and model contents. The *objectives* component describes the model objective and general objectives (e.g., purpose of the model, ease of use, time scale). The *interoperability* component describes what type of interoperability relationship will exist between models (this refers to the IRMs). At this stage, the interoperability types will be determined between Type A.1: General entity transfer and Type B: Resource Sharing. The model objectives will raise the following questions to identify the interoperability type:

- Is there a need to transfer an entity?
- Does the receiving model have any limitation on receiving entities?
- Is there more than one model passing entities?
- Is any resource shared among the models?

- Is any event shared between the models?
- And finally, is any shared data used by the models?

The *inputs* that will feed into the model are determined by the objectives; and *output* is the expected response from a simulation run. Finally, the *model content* describes the range of the model and the level of detail for each component.

3.5.10 Distributed simulation paradigm

The level of abstraction will also be determined during conceptual modelling. The interoperability between models does not determine the level of interaction details. But level of detail does not influence which simulation approach is most appropriate. In Figure 3.7, Borshchev and Filippov (2004) have presented three major approaches used in the simulation industry.

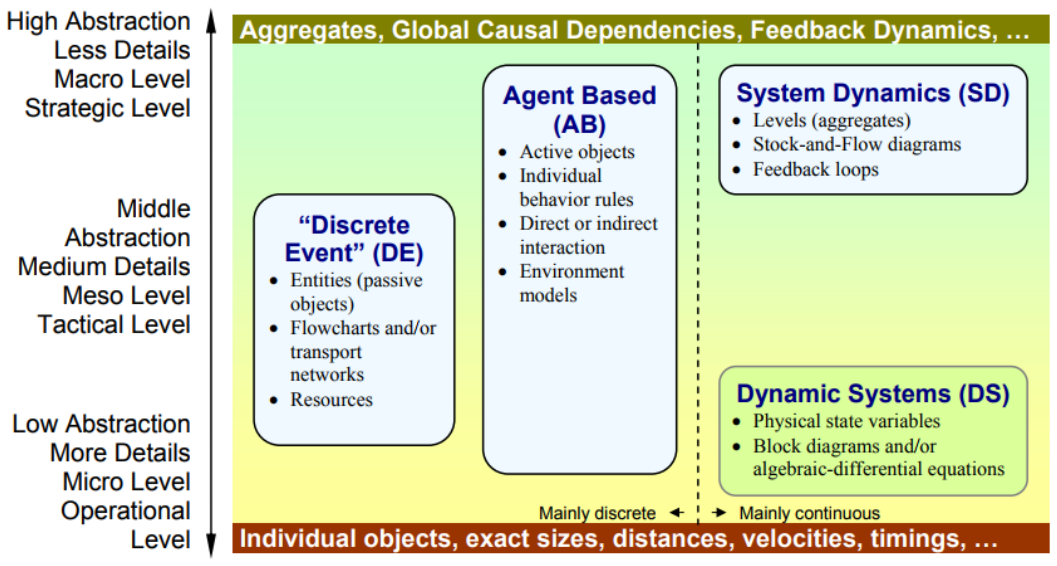


Figure 3.7: Approaches in Simulation Modelling (Borshchev and Filippov, 2004)

In this figure we can see that two of these simulation approaches, i.e., Discrete Event (DE) and Agent Base (AB), are classified under discrete modelling while system dynamics is classified under continuous modelling. In the last few decades, ABS has been used as a purely academic approach. The difference in approach between these systems and the separate practitioner expertise required has restricted the modelling communities' ability to interact with each other (Borshchev and Filippov, 2004). However, industrial requirements have forced practitioners to combine approaches for deeper insight into complex interdependencies (e.g., research done by Anagnostou (2014), that uses Discrete-event and Agent-based distributed simulation modelling). The question, therefore, is not which is the best approach to use, but which approach suits the abstraction requirement.

3.5.10.1 Discrete Event

DE is an approach to simulation that dates back to the 1960s when Geoffrey Gordon at IBM conceived the idea of a General Purpose Simulations System or GPSS (Gordon, 1961). As described in Chapter 2, "in DES the state variable changes only at discrete set points in time" (Banks et al., 2004). This means there is a time interval in which the entity enters or leave an activity. Since this approach has been practiced over a number of decades, there are many COTS tools and packages for use in such industries as production, logistics, defence, health care, and manufacturing. This approach was further enhanced with the introduction of VIS. As discussed in our previous sections, creating a distributed simulation model faces the additional challenge of interoperability. Due to this, it is extremely difficult to capture the differences between all approaches and implementations used across industry. As DES is the most widely used approach, Distributed Discrete Event simulation was selected for this research with stochastic elements.

3.5.10.2 Agent Based

AB is the most recently emerging approach used for modelling deterministic or stochastic simulations. Unlike discrete event and system dynamics, ABS models are decentralised. The system behaviour is not defined globally, instead global behaviour emerges from the individual's simulation or model behaviour. This means that the agents interact with the environment and with other agents, which results in changes of agent behaviours by changing their properties. Because of this behaviour, it is similar to the Object-oriented approach (North and Macal, 2007).

Despite the functional and implementation differences between AB and DE approaches, they are significantly similar to each other, because in both simulation approaches time is progressed in discrete time steps (Pawlaszczyk and Strassburger, 2009). Law's conclusion was that "Agent Based Modelling is just a special case of Discrete Event Simulation" (Law, 2014).

3.5.10.3 System Dynamics

In common with the longevity of the discrete-event approach, System Dynamics dates back to the late 1950s. System Dynamics is "the study of information-feedback characteristics of industrial activity to show how organizational structure, amplification (in policies), and time delays (in decisions and actions) interact to influence the success of the enterprise" (Forrester, 1958). Social, ecological, and urban systems are some examples where system dynamics is used to determine the flow between stocks and information. The mathematical structure of a system dynamics simulation model is based on a system of coupled differential equations (systemdynamics.org, n.a.). This approach seeks to capture the behavioural influence of continuous dynamic endogenous change upon a real world system by progressively selecting narrow discrete intervals of computational time.

3.6 IRMs Objectives

This section aims to describe the simulation environment objects, i.e., the problem statement that is the first step in determining the modelling methodology used. As discussed previously, there are a total of four IRMs with the first, Type A: Entity Transfer, sub-divided into three separate models all related to Entity Transfer issues. Hence, a total of six different problem statements for each model is explained in this section.

- TYPE A: Entity Transfer
  - TYPE A.1: General Entity Transfer
  - TYPE A.2: Bounded Receiving Element

   - TYPE A.3: Multiple Input Prioritisation
- TYPE B: Shared Resources
- TYPE C: Shared Event
- TYPE D: Shared Data Structure

3.6.1 Problem Statement for Type A.1: General Entity Transfer

General Entity Transfer occurs when transferring an entity from one model/federate to another. For example, in Figure 3.8, M1 represents a cookie preparation area and M2 a packing line. In this system, cookies at M1 from queue Q1 are cooked at activity A1, are transferred to M2 arriving at buffer queue Q2, and are then packed at activity A2. For example, if M1 is a firing barrel production unit and M2 is a tank Assembly line, then a firing barrel would leave a finishing activity in M1 at T1 and arrive in a buffer in M2 at T2 to await fixing.

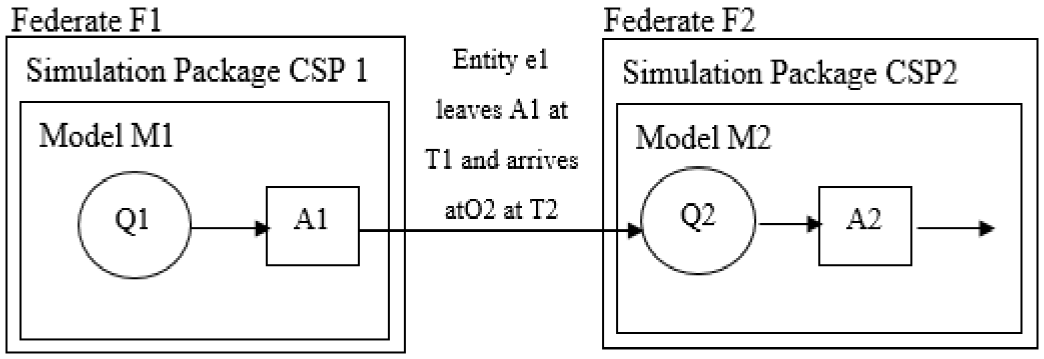


Figure 3.8: Type A.1: General Entity Transfer

3.6.2 Problem Statement for Type A.2: Bounded Receiving Element

The Bounded Receiving Element type is similar to Type A.1 except the receiving buffer Q2 may be bounded, i.e., a buffer with limited space as illustrated in Figure 3.9. In such a case, a check must be performed before sending an entity from M1 to ensure the buffer at M2 has sufficient space and is ready to accept the entity. If there is no space in Q2, then M1 should retain the entity until space is available. For example, if the packing of cookies at M2 is slower than the production of cookies, then the production line at M1 must hold the cookies until space is made available at the packaging buffer Q2 to enable cookie transfer to recommence.

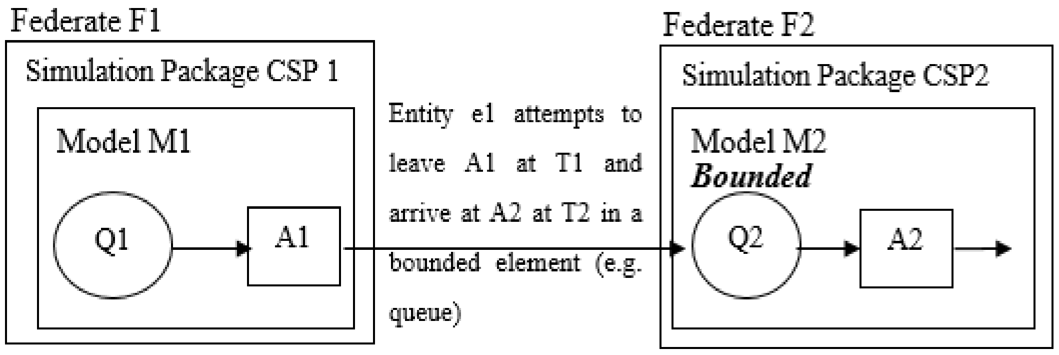


Figure 3.9: IRM Type A. 2: Bounded Receiving Element

IRM Type A.2 – Bounded Receiving Element was originally classified by the CSPI PDG as Type II – Synchronous Entity Passing. A second paper from the PDG (Taylor, 2006a) described its distinction from Type I by explaining the specific difficulties involved in synchronous entity passing and setting out the PDG's proposed solutions.

3.6.3 Problem Statement for Type A.3: Multiple Input Prioritization

This type represents a situation in which a receiving queue in one model can receive entities from more than one model (Figure 3.10). The problem arises when entities from two or more models arrive at the same time. Several strategies are available to resolve such conflicts, for example, orders of priority could be established. In this case, if model M3 is due to receive entities simultaneously from models, M1 and M2, then M3 could set a higher priority for receiving entities from M1. Alternatively, a decision to embed a similar priority within the entity itself could be made at runtime. This IRM does not include cases where the receiving element is bounded as described in IRM Type A.2.

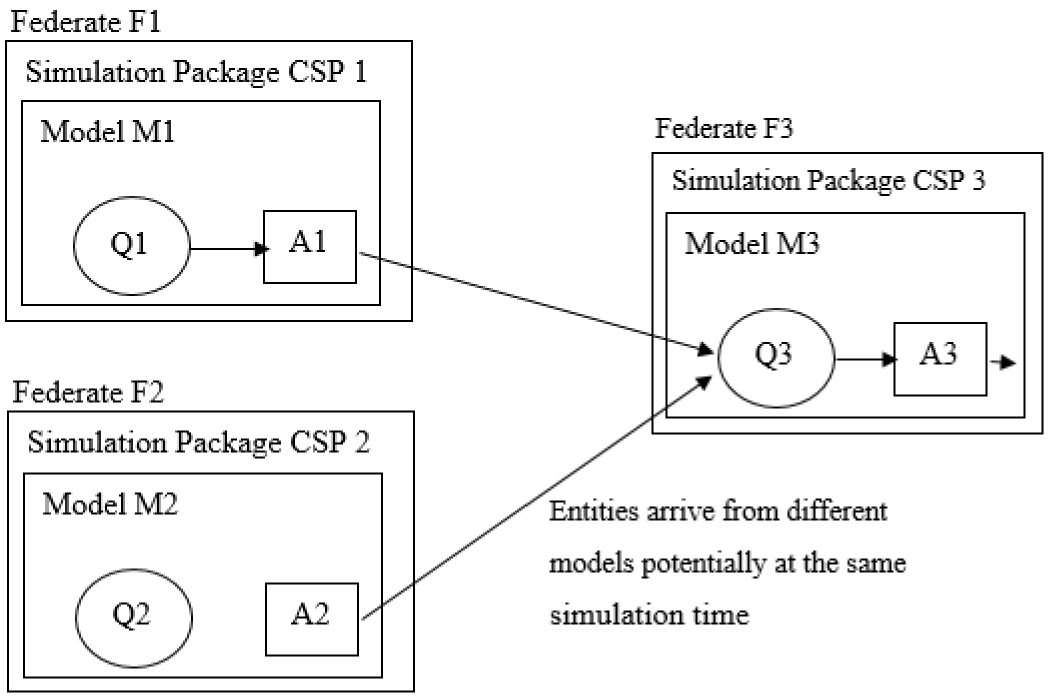


Figure 3.10: IRM Type A.3: Multiple Input Prioritization

3.6.4 Problem Statement for Type B.1: General Shared Resources

The Shared Resources IRM (Figure 3.11) represents a problem related to resource(s) sharing between two or more distributed simulation models. The current state of such resource(s) must be represented consistently in each model that shares the resource(s). For example, in the cookie manufacturing simulation, such resources could be a machine operator and a quality tester.

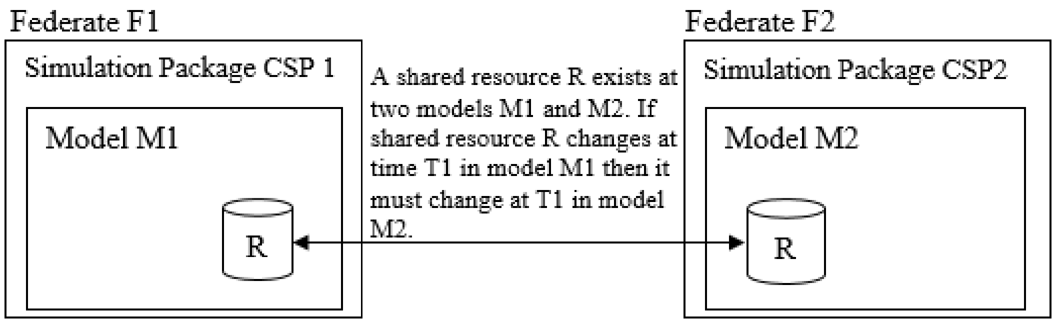

Federate F1
Simulation Package CSP 1
Model M1
R
A shared resource R exists at two models M1 and M2. If shared resource R changes at time T1 in model M1 then it must change at T1 in model M2.
Federate F2
Simulation Package CSP2
Model M2
R

Figure 3.11: IRM Type B.1: General Shared Resource

3.6.5 Problem Statement for Type C.1: General Shared Event

A Shared Event deals with issues in which an event is experienced by two or more models. For example, in Figure 3.12, an event (E) represents a fire alarm sounding requiring both models, M1 and M2, to stop production immediately.

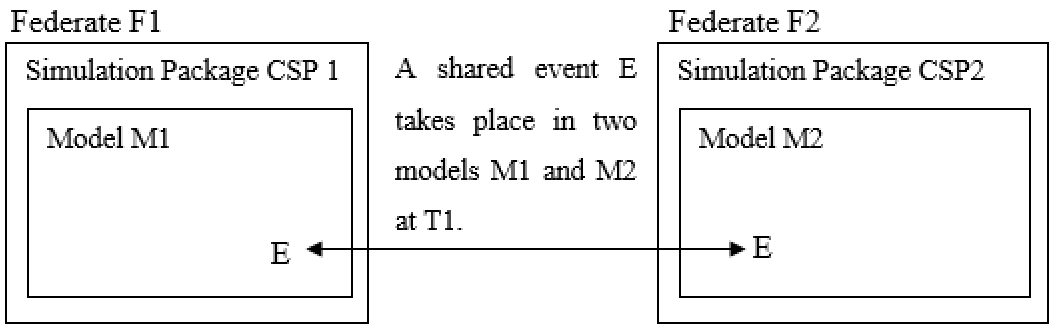


Figure 3.12: IRM Type C.1: General Shared Event

3.6.6 Problem Statement for Type D.1: Shared Data Structure

Shared Data Structure IRM (Figure 3.13) represents problems concerning data sharing between two or more models or federates, in which maintaining vital data consistency can be a challenge.

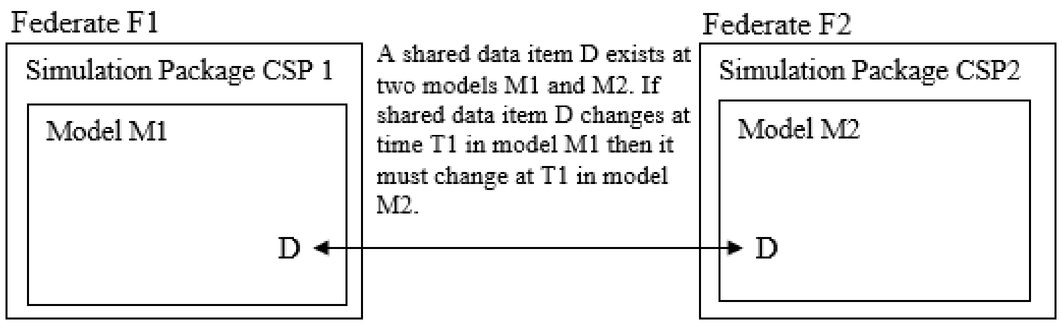

Federate F1
Simulation Package CSP 1
Model M1
D
A shared data item D exists at two models M1 and M2. If shared data item D changes at time T1 in model M1 then it must change at T1 in model M2.
Federate F2
Simulation Package CSP2
Model M2
D

Figure 3.13: IRM Type D.1: Shared Data

3.7 Summary

This chapter first considered the importance of interoperability standards for the modelling and implementation of distributed simulations. Issues involving effective data exchange, communication, and representation were presented, stressing the need for common interpretation. The HLA object model was introduced as a solution operating at abstract level by providing defined interaction and attribute classes. Similarly, the partial HLA solution to ownership and control of data elements was explained as following publish and subscribe methods. It was noted that overcoming semantic and syntactic conflicts was best addressed at the highest abstract level. The importance of time synchronisation between models was introduced with explanations of alternative techniques and the advantages presented by the HLA time management service.

The problem of non-transferable libraries and reference models typically created by COTS software vendors was noted as reinforcing the need for a DSI Framework. The superiority of HLA as a medium for embracing interoperability solutions was stressed given its expanded development from a defence industry utility to a medium actively seeking distributed simulation solutions for wider industry.

The approach to interoperability used in this thesis was explained as aiming to harmonise the common features of different distributed simulation methodologies and bring them together with additional research. The thesis methodology was confirmed as following IEEE standards and the recommended standard practice for DSEEP. The seven DSEEP stages of the development life cycle were noted together with the introduction of a missing stage of V&V for interoperability in distributed simulation after integrating the simulation.

A further modification to the DSEEP model – of introducing the SDEM at conceptual Stage 2 rather than the recommended developmental Stage 4 – was explained as following SISO Conceptual Modelling Group’s belief that interoperability issues should be identified at conceptual modelling level. Importance was thus placed on adopting and modelling the six IRMs from SISO PDG at Stage 2 to identify and manage specific interoperability problems at early analysis stage. Additionally, it was noted that the V&V stage was introduced in this thesis to verify that the assembled distributed simulation system meets required specifications and that it validly represents the real world system.

Application of the seven stages of the DSEEP – from conceptual development to design, execution, validation and verification of the simulation environment – were individually explained in detail with emphasis on the challenges faced.

The chapter continued with a broad discussion on the relationship between conceptual modelling and interoperability, including the balance to be struck between complexity and accuracy. As a result, a revised conceptual modelling mthodology was presented for the use of distributed simulation industy practitioners. Three different simulation approaches – discreet event, agent based, and system dynamics – were also briefly outlined and evaluated. Finally, this chapter concluded with illustrated and explanatory examples of each of the IRM problem types representing known interoperability problems in distributed simulation.

Chapter 4: DSI Framework

4.1 Overview

Earlier chapters have stressed the importance of interoperability between models in a distributed simulation and have explained the function of the RTI in enabling interaction between models. The function of IRMs in identifying known common problems in communication between models during distributed simulation operation has also been discussed. In this chapter, DSI Framework is introduced, which incorporates a new IRM Manager layer into the standard HLA distributed simulation communication structure. After explaining the internal structure of the IRM Manager, the chapter continues with a rationale for the tools chosen for assembling and operating the new DSI Framework, followed by a breakdown of the different forms and sources of data and statistical distribution involved in analysing simulation performance and measuring achievement. Finally, each of the six IRMs installed with the IRM Manager is analysed for its constraints, limitations, and requirements.

4.2 The Framework

Distributed simulation using HLA standard is defined in Figure 2.18 in Chapter 2. There are several other protocols for distributed simulation but the most dominant, popular, and accepted standard is HLA IEEE-1516. The description of HLA in Chapter 2 (Section 2.5.2) explained how individual federates interact with each other using the RTI implementation. The responsibility of RTI and its functions are also explained in Chapter 2 (Section 2.5.3). From previous discussion, we can acknowledge the importance of interoperability issues identified by IRMs, which cannot be ignored while developing any distributed simulation. If these interoperability issues are addressed by using some standard, then more composable and reusable models can be achieved. Therefore, in this research an IRM Manager layer is introduced between the models and RTI. This provides transparency between the model programme and the RTI communication interface. Figure 4.1 illustrates an abstract view of the proposed DSI Framework. The models within the federates connect directly with the IRM managers to send and receive information. The IRM Managers use the FOM declared by the federates to translate data to and from the models.

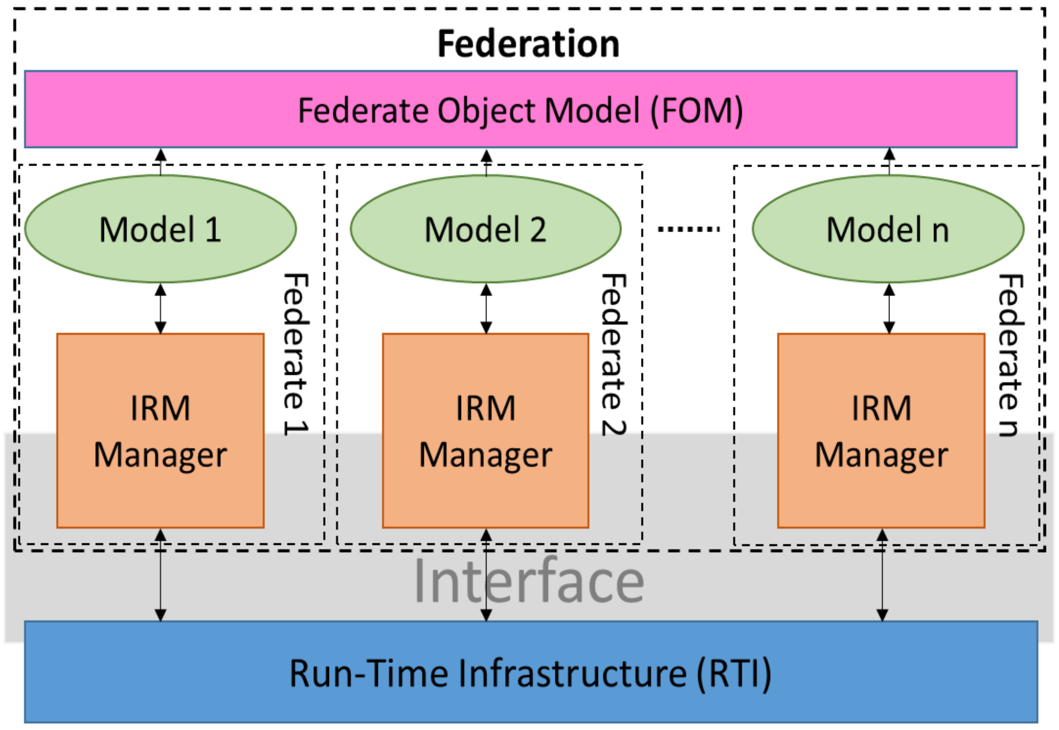


Figure 4.1: An abstract view of Distributed Simulation Interoperability (DSI) Framework

The proposed IRM Manager consists of six components identified in Figure 4.2, i.e., Federate Interface, HLA Interface, Entity Transfer, Shared Resources, Shared Event, and Shared Data Structure. Out of these six components, the first two components – Federate Interface and HLA Interface – are compulsory, while the remaining four are optional as not every federate will use all four components. The selection and use of these components is determined in the conceptual modelling phase where the required type of interoperability between the models is identified. The use of these components could be triggered based on the requirement analysis and an abstract system model.

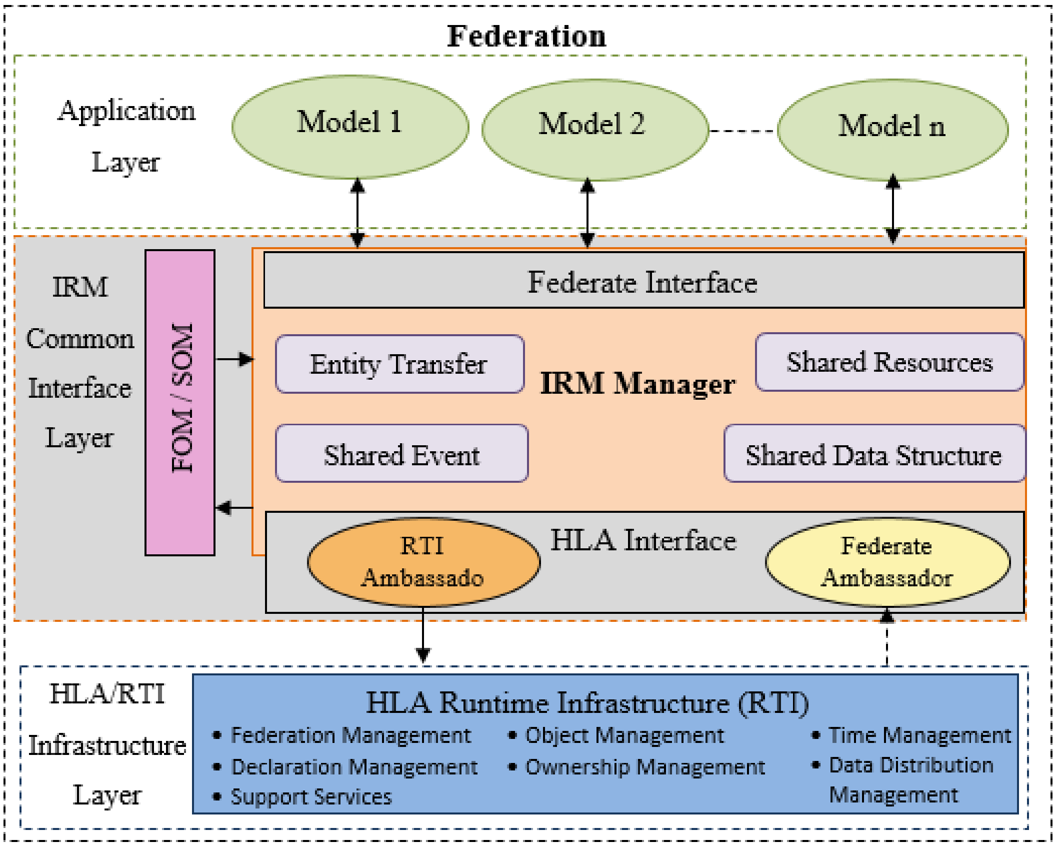


Figure 4.2: Distributed Simulation Interoperability (DSI) Framework

The Federate interface component translates the data objects or variables between the model and the IRM manager. The IRM uses pre-defined FOM objects which might not be syntactically or semantically compatible with the model. For example, a model might save information such as a house door number in actual number but it could also be represented by char or a string in FOM. Similarly, the HLA interface component sends and receives time stamped messages to and from RTI using RTI Ambassador and Federate Ambassador respectively.

The Entity transfer component as defined by the IRM is further classified into three categories all related to entity transfer, i.e., General entity transfer, Bounded receiving element, and Multiple input prioritisation. Again, a federation will not necessarily require the use of all these components. The Shared resource component will be used when a federation requires the use of shared resources. Similarly, Shared event and Shared data structure components will be used when there is a need for shared event or shared data structures to be used by federates in the federation. Different protocols are discussed in this research for each of these four components.

Event synchronisation, data exchange, and time synchronisation between different federates/models is implemented using DSS. A federation is composed of all the federates, which are connected by RTI implementation. Figure 4.1 clearly illustrates the Federation and how each federate is connected with an RTI implementation with the RTI manager providing all the data exchange, time synchronisation and event synchronisation.

4.3 Tools Selection

The rationale behind this research is to find all opportunities to make distributed simulation techniques more flexible, reusable, and composable. Therefore, not only does the technique used for the approach play an important role, but also the tools that are used. The main object of this research is to exemplify the IRMs, therefore the following requirements were identified for selecting the RTI:

- **Platform Independence**: The selected RTI should be generic and able to be used cross-platform.
- **Open Source**: Selecting an open source RTI provides the facility to support continuous research and development with the code able to be modified to meet specific requirements if required. As opposed to commercial software, open source software is cost free and enables access to source code. The other advantage of open source software is the wider community use and support.
- **HLA Evolved support**: The selected RTI must be implemented using the latest standard implementation defined by IEEE 1516e.

- **Backward Compatibility**: The selected RTI should also support the previous versions of IEEE 1516 standard, i.e. RTI 1.3.

The PoRTIco RTI implementation was carefully selected after considering the above requirements. It is cross platform, open source, and fully supports HLA RTI implementation. PoRTIco engages in continuous research and development and is intended to provide a production grade RTI (Portico, 2009a; 2009b) for simulation and training. It is developed and maintained actively by its team and contributors, i.e., Australian Defence Simulation Office (ADSO) and is licensed under the terms of the Common Development and Distributed License (CDDL). Figure 4.3 presents complete PoRTIco Local RTI Component (LRC) architecture.

PoRTIco provides a modular architecture enabling its behaviours to be modified or replaced at five different points. This flexibility makes PoRTIco RTI implementation more desirable for this research. The developer can load their own programme logic at the following five different points.

- **RTI Request:** LRC sends messages to this RTI Request component. This component maintains an “Action Queue” into which new requests can be placed.
- **LRC Request:** message types can be turned into message before sending the request to RTI by using this component. It validates the request on the LRC side.
- **LRC Callback:** This component processes all callbacks from the “LRC Queue”. It also triggers the FederateAmbassador Callbacks.
- **LRC and RTI Connections:** This component works as an RTI communication interface, i.e., it sends and receives information to and from RTI.
- **RTI action:** This component processes the objects in the “Action Queue”. During this processing of objects, a federate callback message might be generated, i.e., if a callback message is required by the process.

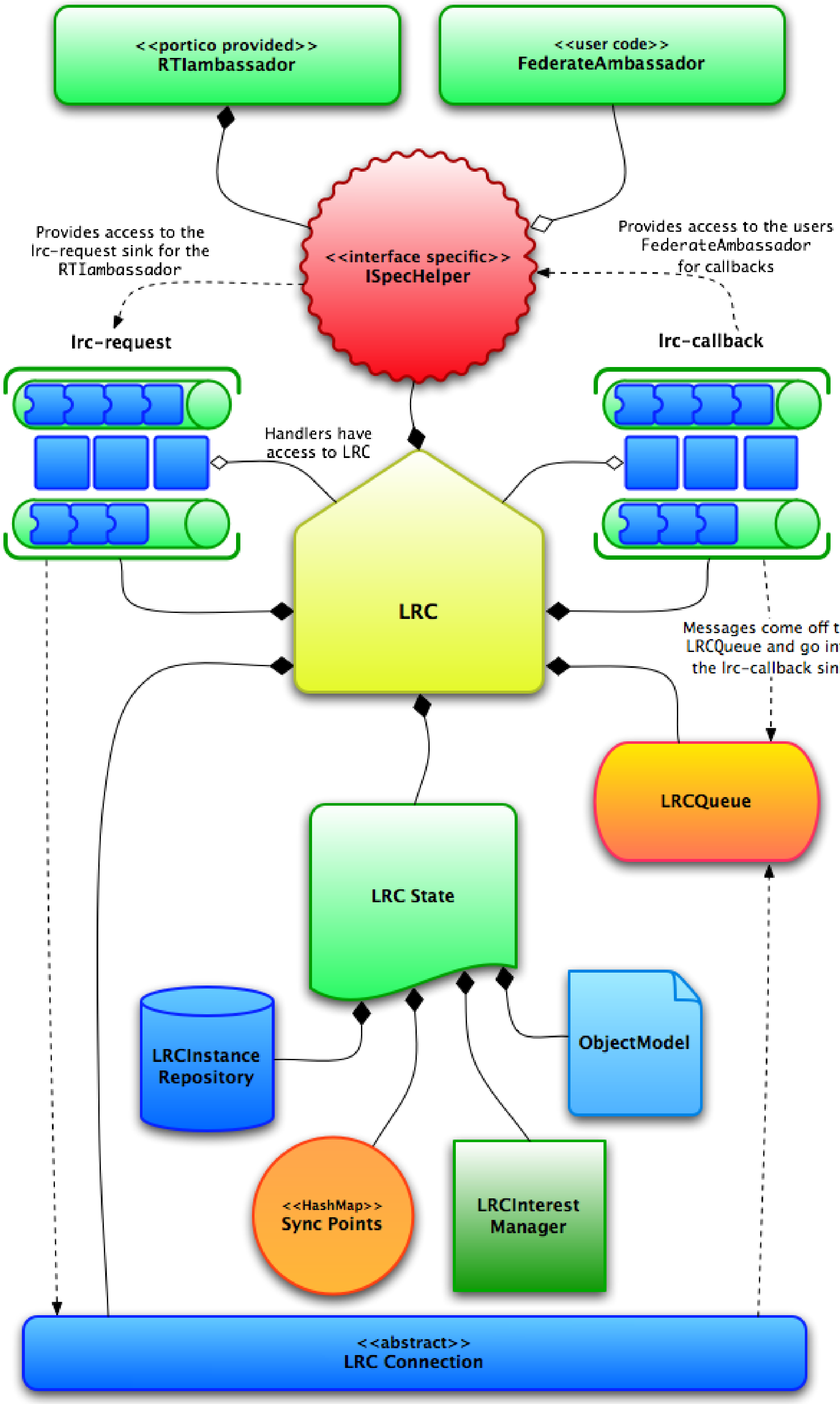


Figure 4.3: LRC Architecture (the poRTIco Project)

However, in this research, for portability and reusability purposes, Pitch pRTI free evaluation version was also used in a few instances. Pitch pRTI is a commercially licenced distributed simulation infrastructure implementation based on HLA IEEE 1516-2010 standard. It provides a good visual implementation for both LAN and Cloud, which can be accessed through a web browser, smartphone, or tablet. Compared to Portico, Pitch pRTI also supports backward compatibility for older versions of HLA implementation. There are a number of additional support modules and tools available, e.g., Pitch visual OMT Editor, Pitch Talk, etc. The RTI is packaged with an additional layer for fault tolerance, robustness, and ease-of-use.

For model implementation, Repast Simphony was used. It is a cross platform, interactive, and tightly integrated Java-based modelling toolkit. This toolkit is available for Microsoft Windows, Linux, and Apple Mac OS X. Repast Simphony was developed by Sallach, Collier, and others (Collier et al., 2003) in 2000 at University of Chicago to support "rapid social science discovery". The objectives of Repast Simphony are as follows:

- A strict separation between model execution, data storage, model visualisation, and model specification.
- User model components based on plain Java objects open to external software for accessing or replacing.
- Tasks automated for model developers.
- "Boilerplate" code can be replaced or eliminated, i.e., the repetitive code used by the models.
- Simple and direct idiomatic code expressions.

  (North et al., 2005)

Although Repast Simphony was intended for Agent-based modelling, it can easily be adapted for DES. The flexibility of this toolkit makes it ideal for this research. Furthermore, it supports ReLogo, Groovy, and "point-and-click" state charts. It also supports Visual interactive (VIS) simulations. It has been successfully used in many industrial scenarios such as hydrogen infrastructure, supply chains, consumer products, traffic systems, and social sciences. Repast has two versions, “Repast Simphony” and “Repast HPC”. Repast HPC, or “Repast for High Performance Computing”, is used for large models to run only on super computer using C++. Repast Simphony is most commonly used as a multipurpose toolkit (North et al., 2013).

Repast Simphony toolkit uses the well-known Eclipse as a primary development tool. Eclipse is open source, free, and a widely used development environment. Eclipse also supports multiple languages including Java and C++. There are other Repast Simphony Eclipse plug-ins that provides perspectives, views, and tools for creating Repast-specific components

4.4 Data Representation and Distribution

In distributed simulation, data representation and distribution means a collection of quantitative data or numbers on various elements as a result of a successful execution of a simulation run. Although data is involved at different stages, such as data exchange during interoperability, internal model data, etc., this topic concerns the data used for analysing simulation performance and measuring achievement. There are several points where data could be collected, for example, breakdown frequencies, departure patterns, arrival patterns, cycle time, communication time, etc. Mostly, the modeller focuses on quantitative data and forgets about the importance of qualitative data (Robinson, 2014). In this research, the data collection and analysis focus on both quantitative and qualitative data.

4.4.1 Data Requirements

Data representation and distribution starts at the identification of data requirements. According to Pidd (2003), data requirements can be split at three different stages in simulation modelling: contextual data, model realisation, and model validation.

4.4.1.1 Contextual Data

Contextual data is basic data, a layout diagram, or the source of a problem experienced when developing thorough understanding of the system at problem identification stage. At this stage, the data is only used for identifying and understanding the problem domain, therefore, large amounts of data collection can be avoided. This data feeds into the conceptual modelling phase and becomes the basis for further data collection. For example, in this research, input data, frequency of message passing, data distribution and simulation performance related data were considered.

4.4.1.2 Model Realisation

This second type of data required is used to determine the development of a computer model, i.e., detailed data is required about entity arrival, frequency of breakdowns, cycle time, frequency of departure, etc. This data provides a basis for developing the computer model, because all data required by individual components of a simulation (including interoperability of all models) is identified at this conceptual modelling stage.

4.4.1.3 Model Validation

Model validation is the final type of data requirement. A simulation project is classed as successful if the simulation model as a whole is correctly representing the real world system with acceptable accuracy. Therefore, in this stage the data from the real world system is compared with the results of the simulation run to validate the model.

4.4.2 Data Procurement

There are following three different types of data identified at three different stages of data requirement described in the previous section.

4.4.2.1 Category A

This category of data is related to information or data previously known before the proposal of a simulation project. For example, cycle time of a production unit and breakdown time of the production unit. This type of data is usually available and is used in the problem identification stage. This data may not be necessarily be collected for the purpose of the simulation and may exist historically for some other purpose.

4.4.2.2 Category B

This category of data needs to be collected as a requirement for the simulation project. This often includes departure patterns, service time, shared resources, time to repair, etc. This data might be collected by human monitoring or other systems. The recommended approach is the direct involvement of the modeller. This will help the modeller to understand the system behaviour at grassroots level.

4.4.2.3 Category C

This category of data is neither directly collectable nor available. This could be as a result of missing real world system data, i.e., for a proposed system. This type of data is assumed to the closest possible known factors or the experience of a domain expert is required to fill the gaps.

4.4.3 System Variability

In a stochastic simulation model, unpredictable or random variability of data is at the centre of the simulation. Many aspects of the simulation are subject to this random behaviour, for example, arrival time, service time, or repair time. The modeller might be aware that customers arrive at random times in the real world, but the exact time can never be identified. Hence, system variability presents a key challenge to meet real world system data requirements. Therefore, statistical and empirical distribution is used to represent system variability.

4.4.4 Empirical distribution

Data from the real world system can be collected that define sequences of events. This could represent the time at which the event occurred. In empirical distribution this predefined collected data is summarised in histogram. During the simulation data values are randomly selected by using random numbers. Mostly the data is provided by the user but the sampling process is hidden. For example, the data might be the customer arriving at a station after every 2, 3, 4 and 5 minutes. The system will randomly select the arrival time and the resultant data could be 3, 2, 2, 5, 3, 4 minutes.

4.4.5 Statistical Distribution

Statistical distributions are defined by some "Probability Density Function (PDF)" or mathematical function. There are many types of distribution each having its own properties and purpose, e.g., Normal, Exponential, Discrete, Erlang, Binomial, Uniform, Poisson, and Triangular distribution. This list is not exhaustive, but this research uses exponential and normal distribution because they are the best known and most commonly used distribution techniques. The reason for using two different distributions is to identify that this research is not limited to any specific distribution method.

4.4.5.1 Normal Distribution

Normal distribution is a commonly used PDF. It is used in the natural and social sciences where the exact value of the data is unknown in the real world system. This distribution technique is used to predict data like customer arriving time, processing time, repair time, etc., in the simulation. Normal distribution is specified with two parameters: standard deviation (σ) and mean (μ) and the PDF for normal distribution (Pooch and Wall, 1993) is:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} exp^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

An example of normal distribution is illustrated in Figure 4.4 where the mean is 2 and the standard deviation is 1. The mean defines the location and the standard deviation defines the spread. The values under the curve are represented by X and the higher the curve, the higher is the probability of that value. This curve is not static, i.e., the centre of the distribution probability can shift left or right and even more flat. This can be demonstrated by changing the mean and standard deviation values as shown in Figure 4.5. The increase in standard deviation value (σ), lowers the centre of the distribution (shown by orange colour) and the change in the mean (μ) changes the position of the centre from right to left shown in green colour.

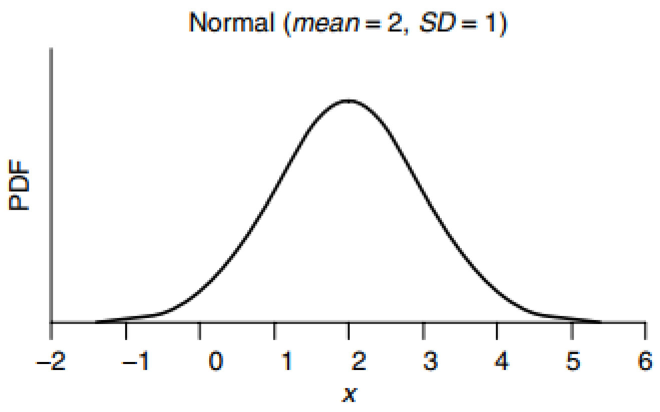


Figure 4.4: Normal Distribution (Robinson, 2014)

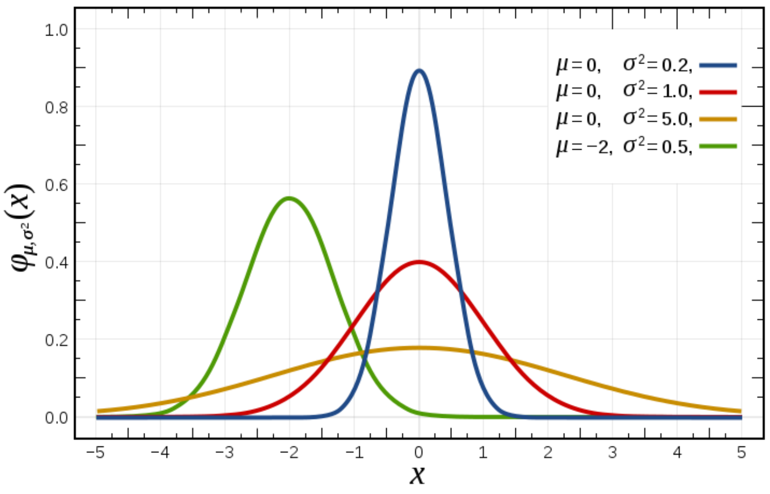


Figure 4.5: Normal density function (summary of Pooch and Wall (1993))

Mostly, in normal distribution the $f(x)$ result can be negative and in decimal value, i.e., the value of $f(x)$ can be less than 0 and it can have n decimal point value, e.g., "1.37", "-0.42". The simulation practice used in this research is conservative and this distribution is used to identify the next event or customer arriving in the queue, then this distribution must produce positive number where $f(x) > 0$. In conservative approach next event can only be generated where next event time (et) is greater than simulation clock(t), i.e.,

$$et >= t$$

Further, in this research the simulation time used is an integer rather than a float (decimal point) therefore the value of $f(x)$ must also be non-decimal, i.e., integer value. Therefore, a round function is used to convert the decimal value of $f(x)$ into plain integer to a nearest level of precision.

4.4.5.2 Exponential Distribution

Exponential distribution is another widely used data distribution in the simulation industry. It defines a process where events occur independently and continuously at a constant average rate. This distribution is also known as negative exponential distribution and is popular in situations like "catastrophic and sudden" change in behaviour, for example, a light bulb suddenly burns out.

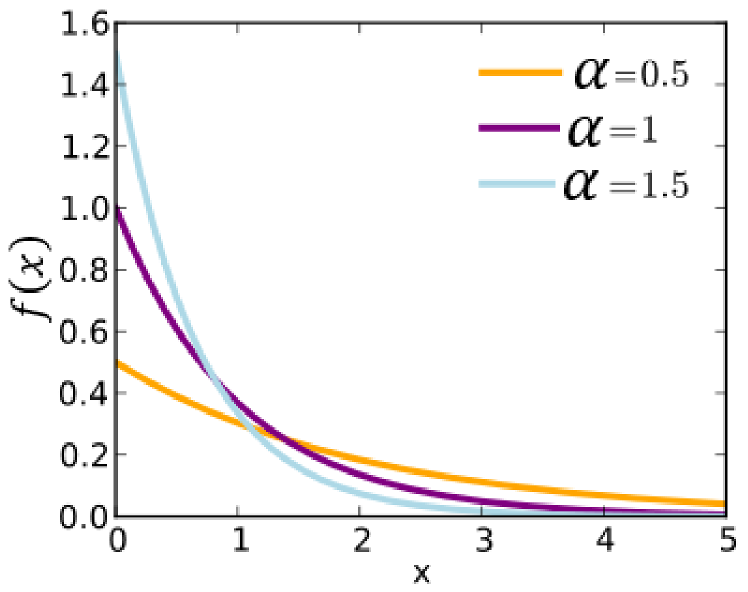


Figure 4.6: Exponential Distribution

The probability function for exponential distribution (Pooch and Wall, 1993) is:

$$f(x) = \alpha e^{-\alpha x}, \quad 0 \leq x < \infty$$

$$f(x) = 0, \qquad elsewhere$$

Where α is a positive parameter, i.e., $\alpha \geq 0$, the standard deviation and mean are the same. The exponential distribution is illustrated in Figure 4.6. Similar changes to the results for exponential distribution were adopted to meet conservative–stochastic simulation modelling, i.e., to ensure the exponential distribution is positive number and values are in integer to the nearest precision.

4.4.6 Data Selection

The IRM solution presented in this research is generic therefore Category A data was not available. All the data used in the conceptual modelling phase is either Category B or Category C, because this framework will be used for the new proposed system. This is one of the main reasons for using statistical distribution. It is also known that the distribution can affect simulation performance, therefore, in such cases multiple distribution values were used to monitor the affects.

4.5 IRM Model Conceptualisation

In this section each IRM (as discussed in the previous chapter) is conceptualised, i.e., requirements, limitations, and constraints are identified. Each IRM is modelled individually therefore the conceptual model presented below for each IRM is totally independent (while applying SISO-STD-006-2010 IRM standards).

4.5.1 Conceptualisation for Type A.1: General Entity Transfer

The problem statement for IRM Type A.1: General Entity Transfer is presented in section 3.6.1. Type A.1 IRM represents a model that interacts on the basis of entities, i.e., models linked together enable one model to pass an entity to another. Type A.1 was previously termed “Asynchronous Entity Passing” in the IRM standard defined in SISO-STD-006-2010. It was called "asynchronous" because there was no direct or immediate feedback when the entity was transferred. All that is required to support the logical link between the two models is the transmission of time stamped entity information between model M1 and model M2 so that model M2 receives the entity information in appropriate order with its own events as illustrated in Figure 3.8. There is no need for a synchronous message exchange between the two models to transfer the entity information (as opposed to the later recommendation for Type A.2). Therefore, in Type A.1 IRM there is also no need for direct feedback when an entity is transferred between models. It is assumed the receiving model can accept the entity in any case because message passing is guaranteed by the RTI.

Entities are not used by all simulation packages, as described in the IRMs. "Entity" in the context of IRM Standard SISO-STD-006-2010 represents some physical or logical system element that is processed by some set of instantaneous state changes at Time T at any given event(s). "Transaction" or "Object" are terms also used in a similar context (Taylor, et al., 2004). Different comments in the literature have been made about the semantics of passing an entity from one model to another (described in Chapter 2). For the purposes of this discussion, "entity transfer" means "the transfer of time stamped information relating to the representation of an object transferred from one model to another" (SISO-STD-006-2010).

Entity transfer representation in this case is kept simple. The bounded receiving element and multiple inputs are not included in the definition of Type A.1. These will be discussed in the following IRM sections. However, there are two conditions defined in Type A.1 standards:

- An entity e1 leaves model M1 from F1 at T1 and arrives at model M2 at F2 at T2.
- Where, T1=<T2 or T1<T2.

4.5.2 Conceptualisation for Type A.2: Bounded Receiving Element

The problem statement for IRM Type A.2: Bounded Receiving Element is presented in section 3.6.2 in detail. Figure 3.9 can illustrates an example where an attempt is made to transfer an entity e1 from Activity A1 at T1 in Model M1 to arrive in bounded queue Q2 at T2 in model M2. When A1 has an entity ready to transfer, it attempts to pass the entity to Q2. When transferring the entity, there are two possible scenarios. If Q2 has space to accommodate the entity, then the entity will be transferred. If Q2 has no space (i.e., Full) then A1 must hold the entity and block its further operations, therefore, any further entity in Q1 must wait for A1 to become available before it can be processed. Eventually, when Q2 has available space to accommodate a new entity, A1 must be notified to transfer the entity to Q2. Importantly, the rest of the model M1 remains functional while A1 is blocked because Q2 is bounded (i.e., the simulation of model M1 should not stop).

IRM Type A.2 represents a model that interacts on the basis of entities passing, i.e., the link between the models is due to the passing of entity to another, similar to IRM Type A.1. However, in this case, the receiving model has a limited element to receive the entities. Unlike IRM Type A.1, IRM Type A.2 needs some sort of synchronised message exchange to transfer entity information between the two models because of the conditions defined above and further detailed below. It is important for both models to send and receive messages to keep the model consistent because, although message passing is guaranteed by the RTI, bounded buffer and blocking behaviour are external to RTI implementation.

The entity transfer representation in this case includes the bounded receiving element along with general entity transfer, but multiple input is not included in the definition of Type A.2. This will be further discussed in later sections about IRMs. However, the following conditions are defined in Type A.2 standards:

- If bounded element Q2 is empty, the entity e can leave element A1 at T1 and arrive at Q2 at T2, or

- If bounded element Q2 is full, the entity e cannot leave element A1 at T1. Element A1 may then block, if appropriate, and must not accept any more entities.
- When bounded element Q2 becomes not full at T3, entity e must leave A1 at T3 and arrive in Q2 at T4; element A1 becomes unblocked and may receive new entities at T3.
- T1 = <T2 AND T3 =< T4
- If element A1 is blocked, then the simulation of model M1 must continue.

Exceptions

- In some special cases, element A1 may represent some real world process that may not need to block.
- If T3<T4 then it may be possible for bounded element Q2 to become full again during the interval if other inputs to Q2 are allowed either locally within the model or externally via other models.

4.5.3 Conceptualisation for Type A.3: Multiple Input Prioritization

The problem statement for IRM Type A.3: Multiple Input Prioritization is presented in section 3.6.3 in more detail. There could be two possibilities for receiving multiple entities by any model element. Consider two Models M1 and M2 capable of sending entities to an element Queue Q3 in Model M3 as illustrated in figure 3.10. If Q3 has implemented First-In-First-Out (FIFO) strategy, then if an entity e1 is sent from M1 at T1 and arrives at Q3 at T2 and an entity e2 is sent from M2 at T1 and also arrives at Q3 at T2, we would expect Queue Q3 order of entity as e1, e2. However, a problem arises when both entities from M1 and M2 arrive at the same time in M3, i.e., where T2=T4. In such circumstances the Queue order could be either e1, e2 or e2, e1, which will be unpredictable.

In some models, a priority order can be specified for such situations, e.g., it can be specified in model M3 that model M1 entities have a higher priority than model M2 in cases where T2 = T4. This priority order could be either specific to a model or could be determined during runtime, i.e., dynamic. Also, priority order could change during simulation if needed.

4.5.4 Conceptualisation for Type B.1: General Shared Resources

Sharing a resource becomes challenging when an activity within a model requires a resource to work on or produce an entity, while an activity from a different model also requires the same resource to work on an entity. In such circumstances the resource will be held up by one of the models until the activity has completed its operations. This means, the other model activity has to wait until the resource becomes available. Similarly, access to a shared resource is not limited to concurrent access only, but also applies to situations where more than one federate share the same resource. Therefore, in a distributed simulation, maintaining the consistent state of that resource in both models becomes a problem.

Figure 3.11 illustrates the state of resource R shared between two models. The IRM Type B.1 General Shared Resource specifies that the known state of the shared resource R must be kept consistent in all sharing models at any given time. For example, if two models M1 and M2 share a Resource R, then both models will have a copy of the resource as RM1 and RM2. The IRM Type B.1 states that the resource state must be guaranteed at any time, e.g., at T1 the state of RM1 from Model M1 will be same as the state of RM2 from Model M2. Additionally, there must be a guarantee that both Models M1 and M2 can attempt to change their copies of Resource R.

The following conditions apply to this IRM to maintain consistency among all copies of shared resources in all participating models.

- If a model M1 wishes to change its copy of R (RM1) at T1 then the state of all other copies of R will be guaranteed to be the same at T1, and

If two or more models wish to change their copies of R at the same time T1, then all copies of R will be guaranteed to be the same at T1.

Strategies defined in IMR A.3 Multiple Input Prioritisation could be adopted if two or more models wish to request a Resource R or wish to change their copies of Resource R at the same time.

4.5.5 Conceptualisation for Type C.1: General Shared Event

The problem statement for IRM Type C.1: General Shared Event is presented in section 3.6.5 above and Figure 3.12. can illustrates an example of a problem where the same event E is shared between two or more models. In the figure, model M1 shares an event E with model M2 at T1. Hence, both models should schedule their local events, i.e., EM1 and EM2, respectively, at the same time T1. There are two guiding conditions for this IRM:

- Both copies of the event, i.e., EM1 and EM2 (in our example), must be guaranteed to occur once the shared event is triggered.
- Both copies of the event, i.e., EM1 and EM2, are instigated at the same time by M1 and M2.

Similarly, if there are more than two models then all events E in all models must take place at the same time T. This IRM is also defined as a guaranteed execution of all shared events in all models at the same time. For example, a situation where a building is on fire. Consider each house in a building is a model and they run their daily life events in their homes, but suddenly a fire alarm is triggered. Every other household member will stop their activities and start the evacuation procedure at the same time and soon after that the building will be empty. In this example, the fire alarm was the trigger to run a shared event “evacuation” at the time it was triggered. Similarly, in an industrial assembly line example, if the packaging machine breaks down, this will cause all other production units to stop as the products cannot process to final stage of packing. Hence, an event is triggered at a certain time when the packaging machine breaks down. A halt message will be sent to all other production units and all units will stop production and execute a halt procedure as soon as the message is received.

There are some other considerations while planning for a shared event. Scheduling of events occurring at the same time might trigger cancellation or changes in an event. In the industrial assembly line example, the production units were scheduled to run an event to produce new entity or a unit. But when the packaging machine breaks down, all future scheduled production unit events must be removed from the next event list. This is further explained in chapter 3 (Section 3.2.2.3) Event Synchronisation. Additionally, these shared events could be bidirectional, i.e., it is not necessary that only one model will generate a shared event trigger. Again, in our industrial assembly line example, if any production unit breaks down, then this will trigger a halt to the packaging unit as it is dependent on production. Also, we cannot exclude the possibility that there will be multiple types of triggers and shared event processes in real practice. Again, in our industrial assembly line example, a factory might have more than one production unit feeding to one packaging unit and if one production unit breaks or slows down with another still working then an event message can be sent to the packaging unit to reduce packaging speed by some factor and / or give information about which unit is broken down, etc.

4.5.6 Conceptualisation for Type D.1: Shared Data Structure

Shared Data Structure could be a transaction record or a variable shared between the models. In a distributed simulation environment, the communication between the models is restricted to the message exchange. Clearly, the distributed models have no physical shared memory and can only communicate via message passing using some sort of underlying technique. This state of disjointedness due to distribution is unnecessarily restrictive from an application modeller viewpoint where the data need to be shared between models. Due to this distributed data structure the data is replicated among the models and much effort is spent on keeping this data consistent by having many specified events. Hence, the immediate challenge in distributed simulation for Shared Data Structure is about consistency of the state of that data structure. However, shared data structure has similar issues to the previous topic of Shared resources, but in simulation, resources are different to data. This IRM also covers issues related to a single data item such as an integer.

Figure 3.13 illustrates the state of shared Data D, shared between two models. The IRM Type D.1 General Shared Data Structure stipulates that the state of the resource R must be kept consistent in all sharing models at any given time. For example, if two models, M1 and M2 share a Data Structure D, then both models will have a copy of the Data as DM1 and DM2. The IRM Type D.1 states that both copies of DM1 and DM2 must be guaranteed to be the same at any given time T, i.e., the state of DM1 from Model M1 will be same as the state of DM2 from Model M2. Additionally, it must be guaranteed that both Models M1 and M2 can attempt to change their copies of Data Structure D.

The following conditions apply to this IRM to maintain consistency among all copies of shared data in all participating models.

- If a model M1 wishes to change its copy of D (DM1) at T1 then the state of all other copies of D will be guaranteed to be the same at T1, and
- If two or more models wish to change their copies of D at the same time T1, then all copies of D will be guaranteed to be the same at T1.

Unlike strategies defined in IRM A.3 Multiple Input Prioritisation, where if two or more models wish to request a Data Structure D or wish to change their copies of D at the same time T, it could be executed but will not guarantee the correctness of data value. Therefore, additional and more complex algorithms are required to identify which update will happen first, especially in the case of a concurrent update.

4.6 Summary

The focus of this chapter has been the introduction of the proposed DSI Framework, its formulation, and the operation of the principle new feature incorporated into DIS – the Interoperability Reference Models Manager.

A diagrammatic and written description of the DIS Framework has detailed how a new IRM Manager layer has been added to the standard HLA distributed simulation configuration. It was explained how the IRM Manager layer sits between the federate models and the HLA/RTI, while the Federate Interface of the IRM Manager translates data objects and variables from the models into the more abstract language of HLA. Similarly, the HLA Interface of the IRM Manager has been explained as sending and receiving timestamped messages to and from RTI via an RTI Ambassador and a Federate Ambassador applicable to each federal model in the distributed federation.

This chapter next addressed the choice of modelling and communication tools used with the proposed DIS Framework. It was explained that the rationale was to select products that assisted the modeller in implementing flexible, reusable, and composable distributed simulation techniques. On this basis, Repast Simphony was selected for model implementation and PoRTIco as principle RTI.

Next, a listing and description was provided of various types and sources of data that may be required or collected for analysing performance and measuring success after a simulation run. Various systems of statistical distribution used in the simulation industry were also discussed.

Finally, this chapter provided a problem statement for each of the six individual IRMs controlled by the IRM Manager in the DSI Framework. Conceptualising each IRM identified the interoperation problem that exists in each case and the requirements that need to be fulfilled to overcome the problem, together with any limitations or constraints.

Chapter 5: IRM Protocols

5.1 Overview

Discussion of the structure and operation of the DSI Framework with IRM Manager in the previous chapter was followed by problem statements describing communication issues represented by each of the six identified IRMs. In this chapter, proposed protocols for running tests of the DSI Framework are first described beginning with the need for independently developed, generic distributed models representing these problem scenarios. Next, the use of software tools to develop the models is explained followed by justification for the versions of HLA and poRTIco chosen for development, with particular reference to their integration. These introductory factors are followed by detailed descriptions of approaches proposed for developing the simulation models. These cover all six of the IRM Types and sub-Types. A total of seventeen proposals in total were developed to address these six IRMs.

5.2 Proposed Protocols

IRMs define a set of interoperability problems while creating a distributed simulation. As described in earlier chapters, these IRMs consists of four main models. The first model is further sub-divided into three models. Generic, reusable, and discrete event distributed models are developed not only to examine performance and limitations, but also to provide different methods to approach the same problems and their effects. Each IRM model is developed independently in accordance with SISO-STD-006-2010 standard definitions and will be explained in detail further in this chapter. Having these models developed independently allows a deeper understanding of their impact on simulation and how effectively the HLA standard can be utilised to adopt the appropriate approach. The list is not exhaustive, but enough representations are made to give a clear understanding about the challenges. These model approaches can easily be adopted by any simulation modeller to address any of the Interoperability issues (identified by SISO-STD-006-2010) facing distributed simulation models. In the given models, most focus is given to interoperability while keeping the model simple.

The software tools used for developing these models were discussed in Chapter 4, i.e., all models were developed in the Repast Simphony simulator. Although the Repast Simphony is an ABS toolkit, its main simulation engine is based on *Schedule Class* which is a discrete-event engine. Similarly, in the majority, PoRTIco RTI implementation was used as a middleware, but at some places Pitch RTI was also used. Both the toolkit and PoRTIco are open source while Pitch is a commercially licenced RTI. PoRTIco and Pitch both support RTI implementation and Java and C++ programming interfaces, but Repast Simphony includes a Java application programming interface. Hence, the selection of programming language was derived from the Repast Simphony. The selection of Repast Simphony was inspired because it not only has the ability for ABS and DES distributed simulation support, but also provides VIS support tools, which are easy to integrate. The Java programme language interface was also selected because it offers the guarantee to be useable from any ware at any time, i.e., "write once and run anywhere". All the protocols were implemented using Time Advance Request (TAR) approach.

The research began with older versions of PoRTIco V1.0 which implemented HLA 1.3. Later a newer version of PoRTIco V2.1.0 was used to develop the models. This new version of PoRTIco implements HLA Evolve standard RTI. This also gave an opportunity to investigate the performance differences between the old standard HLA 1.3 and the new standard HLA Evolved.

5.2.1 HLA Selection

As discussed in the last section, the research began with HLA 1.3, an old RTI implementation based on IEEE 1516-2000 standard. This standard was first published in 1998 but was first implemented in 2001. The standard was revised a decade later in 2010, focusing on more web supported services and changes in OMT structure. The new IEEE standard moved a step forward toward better reusable implementation. The first PoRTIco implementation of this new standard was seen around 2012, but this was also not complete, i.e., some features like MOM implementation were missing.

The current version of PoRTIco also supports both HLA 1.3 and HLA Evolved implementation. Unlike MÄK and Pitch Commercial RTIs, PoRTIco is easy to use and apply. With the author experience PoRTIco can be the best learning platform because the package comes with no or very little configuration requirements, i.e., plug-and-play. For example, there is no need to configure Central RTI Component (CRC) and LRC in PoRTIco unless the modeller has specific requirements, i.e., running the simulation over a cloud, multiple domains, or complex network configurations. Both the CRC and LRC are created at runtime and are destroyed automatically once the simulation end. The CRC and LRC are designed to identify and establish a network connection on any regular network. An exception to a firewall may be required if sub-domains are involved.

Some of the RTIs needs to configure some of their components before use i.e. CRC and LRC. In some cases, both of the components are always running as a background process. Moving down to the complex networked environment, it is easier to configure these commercial RTI’s because they provide a good user friendly Graphical User Interface (GUI). On the other hand, configuring PoRTIco on a complex network will require higher understanding of network configurations, because this is achieved either through the programming interface or by changing the configuration files. The PoRTIco network configuration was kept similar to the old version, but the newer version is restricted to be used with Java 1.7+ versions.

During the initial published research (Nouman et al., 2013) a distributed hybrid agent-based and DES was developed, using both HLA 1.3 and HLA Evolved standards. The research had two main objectives. The first objective was to demonstrate the interoperability of different simulation techniques, i.e., ABS and DES applying the SISO-STD-006-2010 standard. The second objective was to test the performance of HLA 1.3 over HLA Evolved. Figure 5.1, illustrates the test results from different simulation runs. These experiments clearly identified that running different simulation models on a single PC even with multi core and multi-threading is not advantageous over running the model on a network.

It was also noted that single PC performance runs were more linear with the old HLA 1.3 implementation while the newer version HLA Evolved had an upward "Zigzag" behaviour. But this behaviour was linear when the same simulation run was performed over a network. The performance of HLA Evolved got better as the simulation models increased, but the performance of the new standard was never noted to be any less than the old standard. It was concluded that further research would be done using the new HLA Evolve standards. In the present research, no further models were developed using the older version of HLA.

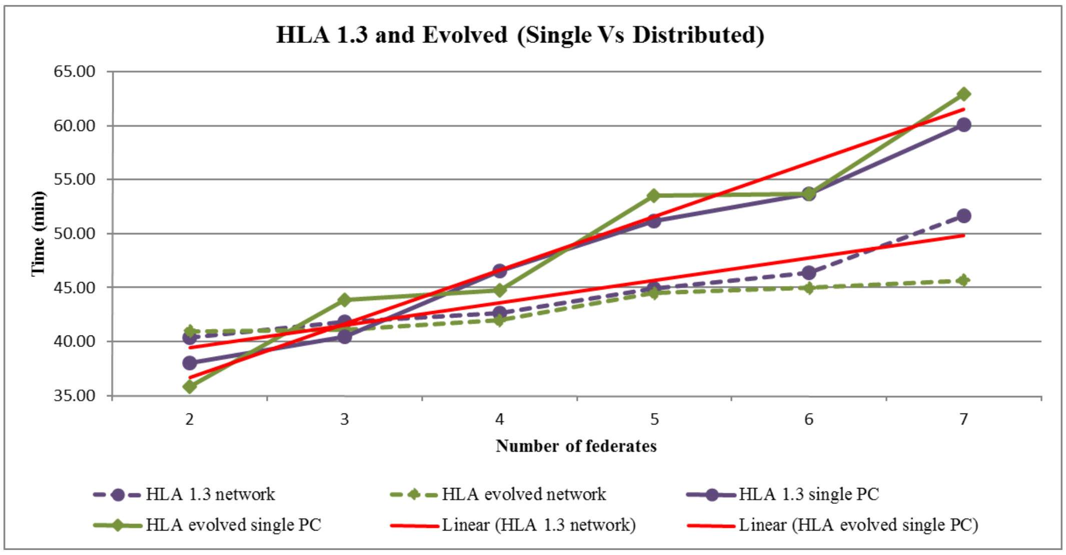


Figure 5.1: Comparison of execution time on a single PC and a network with different numbers of federation models (Nouman et al., 2013)

5.2.2 General Entity Transfer

IRM Type A.1: General Entity Transfer represents an interoperability problem that occurs when transferring an entity from one model to another. This problem is already defined in Chapter 3 & 4 in more detail, but this section will focus on the conceptual modelling of each protocol used to address this IRM issue.

5.2.2.1 Entity Transfer with Interaction Class Protocol

The first possible approach to address IRM Type A.1 General Entity Transfer is by using interaction classes. The relationship between the Models, CSP, and the RTI is shown in Figure 5.2 below. For the purpose of understanding the entity transfer concept, it is necessary to identify the source and destination model, i.e., which model will send an entity and which model will receive it. There could be alternative possible entity routings defined by the models between them. Therefore, it will be assumed that M1 is a source model from which an entity leaves and M2 a destination model at which the entity arrives. Time will be defined as the time when an entity leaves a source model and instantaneously arrives at the destination mode l, i.e., an event happens in M1 at time "T1" marking the departure of an entity from M1 and instantly arriving at M2 at "T2" (where T1=T2=t).

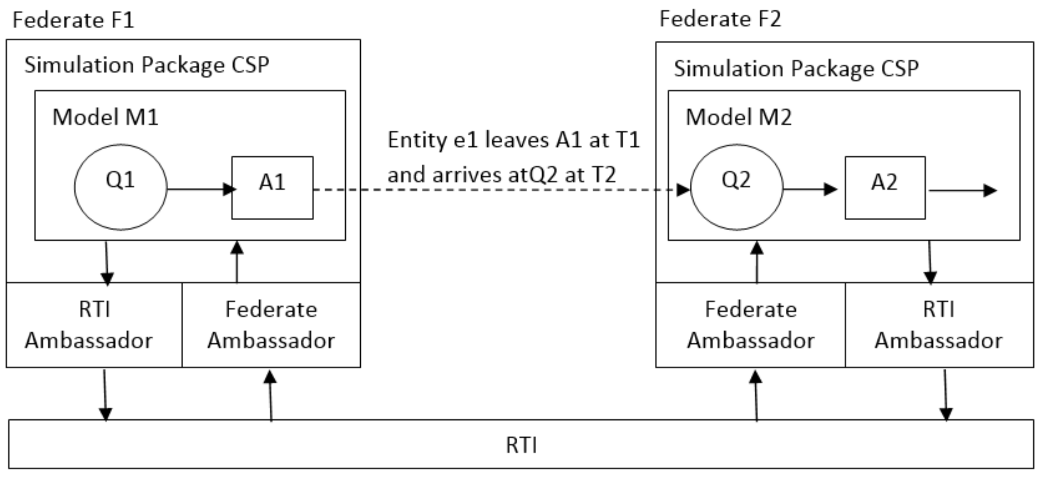


Figure 5.2: General Entity Transfer using HLA

With regard to transferring an entity from a model M1 in federate F1 to a model M2 in federate F2, the entity could be represented as an object with a number of attributes defined by the type of entity. The actual entity representation depends on the model implemented in a CSP and how the modeller will use the entity. For the purposes of understanding, in this research entity is represented as a single interaction class parameter "entity" in both the models. As there are only two models in a given scenario, and the source and destinations are defined, we do not need to define the destination and source names, this will be taken care of by RTI features of Publishing and Subscribing classes. However, it is important in the case of multiple destination federates to either include a destination variable or better to have an entity as an object and define the destination federate as an attribute value within the object.

As defined in IEEE 1516.2-2010 standard for Object Modelling Template (OMT) specification, an interaction class must indicate whether it is published or subscribing. For FOM, valid entries are Publish(P), Subscribe(S), PublishSubscribe(PS), or Neither (N), defined as follows:

- **Publish:** The federate is capable of publishing the interaction class.
- **Subscribe:** The federate is capable of subscribing to the interaction class.
- **PublishSubscribe:** The federate is capable of publishing and subscribing to the interaction class.
- **Neither:** The federate is incapable of either publishing or subscribing to the interaction class.

In the scenario shown in Figure 5.2, the FOM will define the interaction class with the “entity” parameter available for both federates to Publish and Subscribe. Therefore, Federate F1 will publish the “entity” interaction class parameter while Federate F2 will subscribe to the “entity” interaction class parameter. It is important to note that according to the OMT specification “HLAinteractionRoot” remains the superior class over all other interaction classes in Federate Object Model (FOM) and is mandatory. The following is the FOM used to define the interaction class:

```
<interactions>
    <interactionClass>
        <name>HLAinteractionRoot</name>
        <sharing>PublishSubscribe</sharing>
        <dimensions>NA</dimensions>
        <transportation>HLAreliable</transportation>
        <order>Receive</order>
        <interactionClass>
            <name>EntityTransfer</name>
            <sharing>PublishSubscribe</sharing>
            <transportation>HLAreliable</transportation>
            <order>TimeStamp</order>
            <parameter>
                <name>entity</name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
        </interactionClass>
    </interactionClass>
</interactions>
```

In this scenario, Model M1 will produce an entity at time T that needs to be transferred to Model M2. Time to transfer is identified at run time, i.e., it is not fixed because as discussed in Chapter 4 this simulation model uses exponential distribution and all the models are based on a stochastic approach. Entity arriving in Model M1 is generated using an exponential distribution where α=1.5. Similarly, the service time for Activity A1 in Model M1 and Activity A2 in Model M2 also uses the exponential distribution where α=1. Also, this scenario uses zero lookahead, and it assumes that entity travel time from one model to another is zero. Figure 5.3 is a sequence diagram of the proposed approach that follows based on the protocol explained above and Figure 5.4 describe the flow of the programme algorithm.

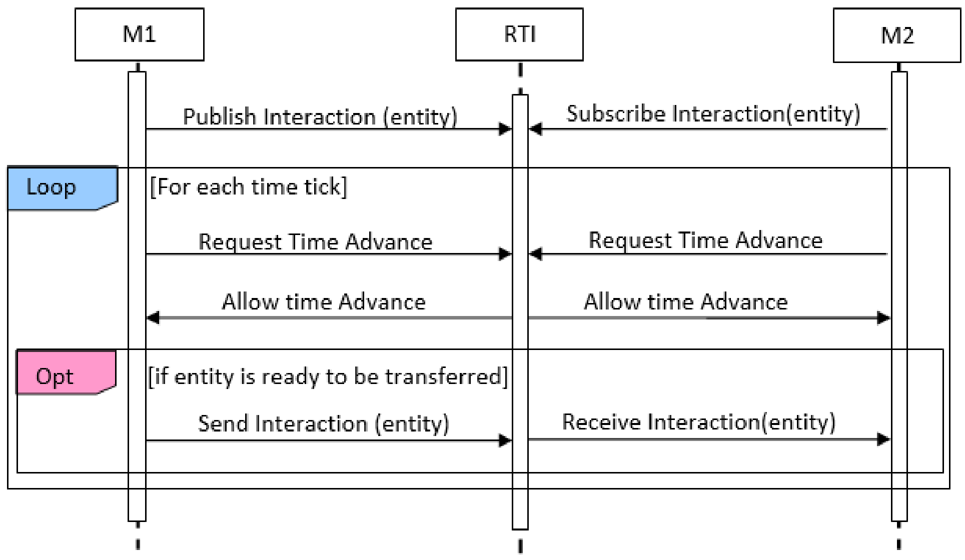

M1
RTI
M2
Publish Interaction (entity)
Subscribe Interaction(entity)
Loop
[For each time tick]
Request Time Advance
Request Time Advance
Allow time Advance
Allow time Advance
Opt
[if entity is ready to be transferred]
Send Interaction (entity)
Receive Interaction(entity)

Figure 5.3: Sequence diagram (SD) General Entity Transfer via interaction using RTI

Sequence of Events

- **At initialisation stage**: Federate F1 will indicate that it is capable of sending entities by publishing entity interaction class parameter to any destination federate who is subscribing entity interaction class parameter.
- **At initialisation stage**: Federate F2 will indicate that it is capable of receiving entities by subscribing to entity interaction class parameter from any source federate who is publishing entity interaction class parameter.
- **Simulation Runtime:** Both Federates F1 and F2 will progress time "t" using the time management service.
- **Simulation Runtime:** At a particular time "Tx" the activity A1 in Model M1 will have an entity to send to Q2 in model M2. Therefore, Federate F1 will send an interaction message with the entity to Federate F2 using the RTI ambassador.
- **Simulation Runtime:** At time "Tx" the interaction message will be received by Federate F2 and will be sent to Q2 in Model M2 via the Federate Ambassador.

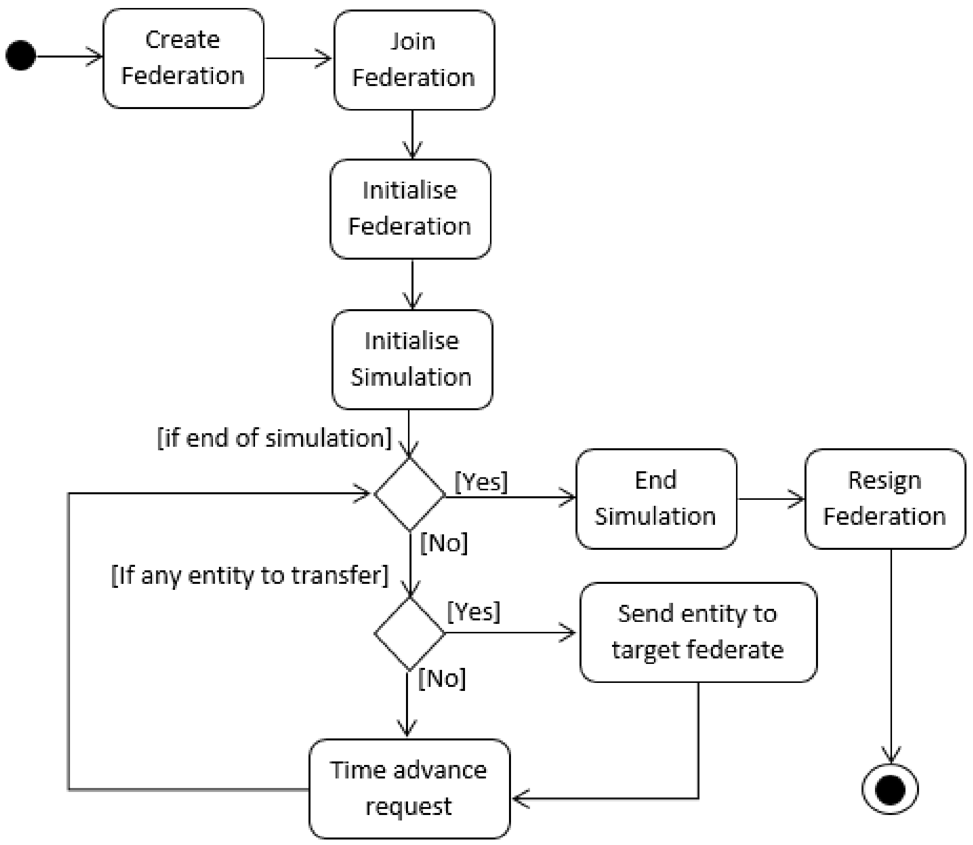


Figure 5.4 : Activity diagram for entity transfer

5.2.2.2 Entity Transfer with Attribute Class Protocol

The second possible approach to address IRM Type A.1 general entity transfer is by using attribute classes. This initial research will indicate if there are any performance issues using these two different approaches. The relationship between the Models, CSP, and RTI is shown in Figure 5.2 above. For the purpose of understanding the entity transfer concept, it is necessary to identify the source and destination model. Therefore, it will be again assumed that M1 is a source model from which an entity leaves and M2 a destination model at which the entity arrives. Similarly, time will be defined as the time when an entity exits from a model and immediately arrives at the destination model, i.e., an event happens in M1 at time "T1" marking the departure of an entity from M1 and instantly arriving at M2 at "T2" (where T1=T2=t).

In regard to transferring an entity from a model M1 in Federate F1 to a model M2 in Federate F2, the entity could be represented as an object with a number of attributes defined by the type of entity. In this research, "entity" represents a single object class attribute in both the models. As there are only two models in a given scenario and also the source and destinations are defined, therefore, we do not need to define the destination and source names, this will be taken care of by RTI features of Publishing and Subscribing classes.

As defined in IEEE 1516.2 2010 standard for Object Modelling Template (OMT) specification, object class must also indicate whether it is published or subscribing. For FOM, valid entries are Publish(P), Subscribe(S), PublishSubscribe(PS), or Neither (N), as described in the last section.

In this scenario, the FOM will define the object class with the "entity" attribute that is available for both federates to Publish and Subscribe. Therefore, federate F1 will publish the "entity" object class attribute while the federate F2 will subscribe to the "entity" object class attribute. It is important to note that according to the OMT specification "HLAobjectRoot" remains the superclass over other object classes in Federate Object Model (FOM) and is mandatory. The following is the FOM used to define the interaction class.

```
<objects>
    <objectClass>
    <name>HLAobjectRoot</name>
    <sharing>Neither </sharing>
    <objectClass>
        <name> EntityTransfer</name>
        <sharing>PublishSubscribe</sharing>
        <attribute>
            <name>entity</name>
            <dataType>HLAinteger32BE </dataType>
            <updateType>Conditional </updateType>
            <ownership>NoTransfer</ownership>
            <sharing>PublishSubscribe</sharing>
            <dimensions>NA </dimensions>
            <transportation>HLAreliable</transportation>
            <order>TimeStamp</order>
        </attribute>
    </objectClass>
</objects>
```

This scenario is similar to the one defined in 5.2.2.1 Entity Transfer with Interaction Classes, with the exception of using the object class instead of interaction class. Therefore, Model M1 will produce an entity at time T that needs to be transferred to Model M2. Time to transfer is identified at run time because it uses exponential distribution. The entity arriving in Model M1 is generated using an exponential distribution where α=1.5. Similarly, the service time for Activity A1 in Model M1 and Activity A2 in Model M2 also uses exponential distribution where α=1. Further, this scenario uses zero lookahead and assumes that entity travel time from one model to another is zero. Figure 5.5 illustrates the sequence diagram for this proposed approach. The sequence of events also remains similar to the illustration in the last section also Figure 5.4 describe the flow of the programme algorithm.

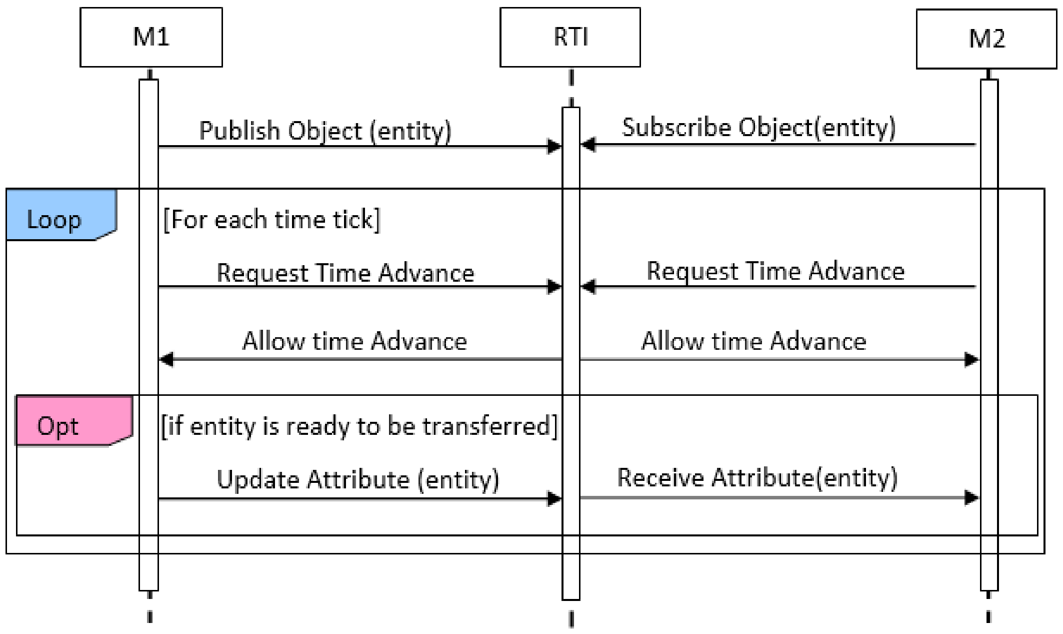


Figure 5.5: SD General Entity Transfer via attribute using RTI

Sequence of Events

- **At initialisation stage**: Federate F1 will indicate that it is capable of sending entities by publishing the entity attribute of an object class to any destination federate who is subscribed to the entity attribute.
- **At initialisation stage**: Federate F2 will indicate that it is capable of receiving entities by subscribing to the entity attribute of an Object class from any source federate who is publishing entity attributes.
- **Simulation Runtime:** Both Federates F1 and F2 will progress time "t" using the time management service.
- **Simulation Runtime:** At a particular time "Tx", the activity A1 in Model M1 will have an entity to send to Q2 in Model M2. Therefore, Federate F1 will send an attribute update message with the entity to Federate F2 using the RTI Ambassador.
- **Simulation Runtime:** At time "Tx" the attribute update message will be received by Federate F2 and will be sent to Q2 in Model M2 via the Federate Ambassador.

5.2.2.3 Entity Transfer using null message Protocol

Chandy-Misra presented a conservative algorithm for distributed simulation and proposed to use null messages to avoid deadlooks (Chandy and Misra, 1979). This approach is further investigated by the similar implementation. The above two different protocols were developed using limited message passing while two similar protocols were developed using a null message technique. These protocols are the same as defined above, but with one exception, i.e., the use of null message. Therefore, the relationship between the Models, CSP and RTI remains the same as shown in Figure 5.2 above. Similarly, the source model and the destination models are the same. The FOM for both the models remains unchanged, i.e., the same FOM is used as defined in the above two approaches. The same distribution will be used and entity transfer will also be instantaneous. The sequence of events, however, is slightly different to the above two protocols. This is illustrated in Figure 5.6 below. Also Figure 5.7 illustrate the flow of the protocol algorithm.

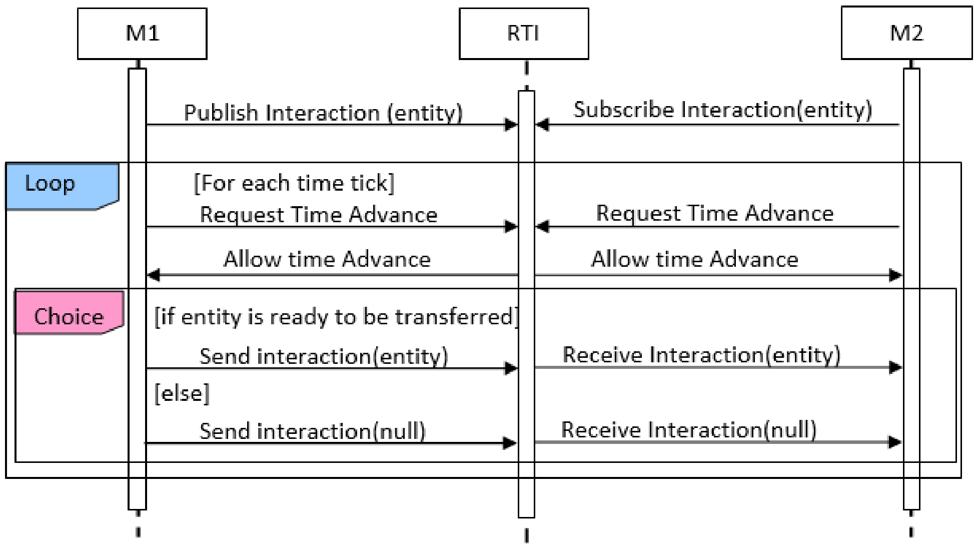


Figure 5.6: Sequence diagram General Entity Transfer using null message

Sequence of Events

- **At initialisation stage**: Federate F1 will indicate that it is capable of sending entities by publishing an entity interaction class parameter (or object class attribute in the case of attribute protocol) to any destination federate that is subscribing to the entity interaction class parameter.

- **At initialisation stage**: Federate F2 will indicate that it is capable of receiving entities by subscribing to the entity interaction class parameter (or object class attribute in the case of attribute protocol) from any source federate that is publishing the entity interaction class parameter.
- **Simulation Runtime:** Both Federates F1 and F2 will progress time "t" using the time management service.
- **Simulation Runtime:** At a particular time "Tx", the activity A1 in Model M1 will have an entity to send to Q2 in model M2. Therefore, Federate F1 will send an interaction message (or attribute update) with the entity to Federate F2 using the RTI Ambassador. If there is no entity to transfer a null message "" denoted by a value Zero 0 is sent.
- **Simulation Runtime:** At time "Tx", the interaction message (or attribute update) will be received by the Federate F2 and will be sent to Q2 in Model M2 via Federate Ambassador. If there is no entity to transfer a null message is received which is ignored by Model M2.

5.2.2.4 Model Verification and Validation

IRM Type A.1 defined the following conditions to satisfy the requirements to achieve the General entity transfer.

- An entity e1 leaves model M1 at T1 from F1 and arrives at model M2 at T2 at F2.
- Where, T1=<T2 or T1<T2.

The above models were programmatically checked against the requirements, and the behaviour of the models remained the same as running a standalone simulation. This is further analysed in the next chapter within the discussion of simulation test results.

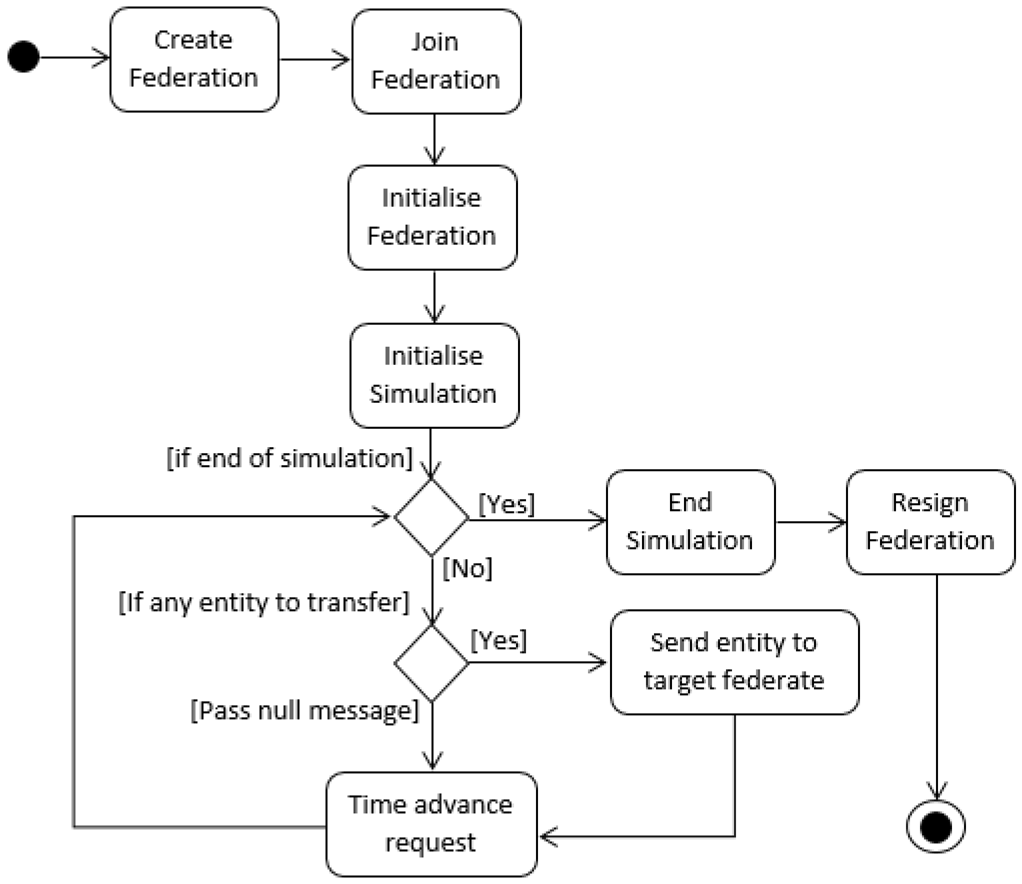


Figure 5.7: Activity diagram using null messages

5.2.3 Bounded Receiving Element

IRM Type A.2: Bounded Receiving Element represents an interoperability problem that occurs in transferring an entity from one model to another model when the initial receiving element for the entity to be transferred is a buffer in the destination model. This problem is defined in more detail in Chapter 4. This section will focus on the conceptual modelling of each protocol used to address this IRM issue. The following three sections consider the three protocols identified to address this problem.

5.2.3.1 Bounded Receiving Element using Queue Update Protocol

The first possible approach to addressing IRM Type A.2: Bounded Receiving Element is by using the simple approach of Queue update. The relationship between the Models, CSP, and RTI is shown in Figure 5.8 below. For the purpose of understanding the entity transfer concept for IRM Type A.2, it is necessary to identify the source and destination model, i.e., which model will send an entity and which model will receive it, and also in this case which model has a bounded receiving queue. There may be alternative entity routings defined between the models themselves. Therefore, it will be assumed that Federate F1 has a source model M1. It is the model from which an entity leaves, and Federate F2 implements a destination model M2 at which the entity will arrive. The queue Q2 in Model M2 is declared as a Bounded Receiving Element. Time will be defined as the time when an entity leaves a source model and may instantaneously arrive at the destination model, i.e., an event happens in M1 at time "T1" marking the departure of an entity from M1 and instantly arriving at M2 at "T2" (where T1=T2=t).

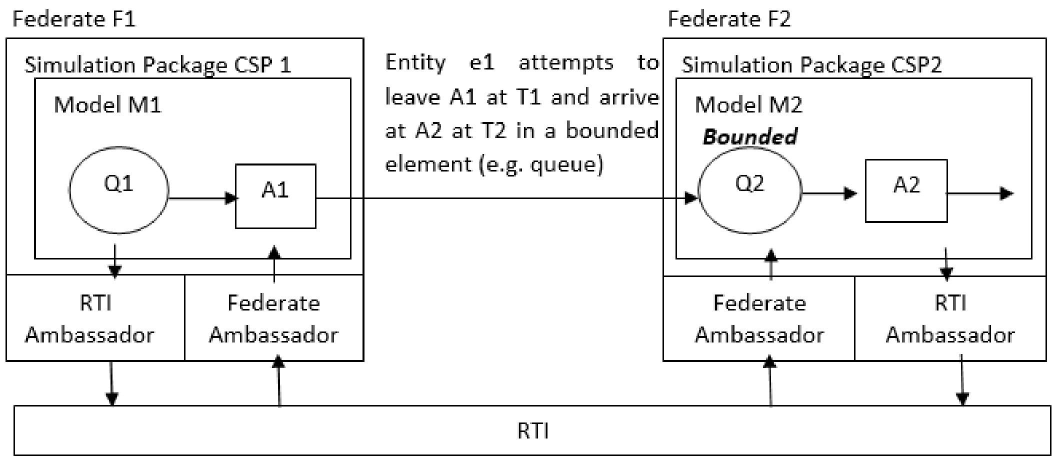


Figure 5.8: Bounded Receiving Element using HLA

Transferring an entity from Model M1 in Federate F1 to a Model M2 in Federate F2, the entity could be represented as an object with a number of attributes defined by the type of entity. Once again, how an entity is actually represented depends on the model implemented by a CSP and how the modeller will use the entity. For the purpose of understanding entity in this research, represents a single interaction class with two parameters "*entity*" and "*Queuestatus*" in both models. As there are only two models in a given scenario and the source and destinations are defined, we do not need to define the destination and source names, this will be taken care by RTI features of *Publish*ing and *Subscribing classes*. However, this is important in the case of multiple destination federates, and it is also important to either include a destination variable, or better, to have an entity as an object and define the destination federate as an attribute value within the object. In short, further attributes could be adjusted based on the simulation requirements and the nature of the entity.

As defined in IEEE 1516.2-2010 standard for Object Modelling Template (OMT) specification, interaction class must indicate whether it is publishing or subscribing. For FOM, valid entries are Publish(P), Subscribe(S), PublishSubscribe(PS), and Neither (N). Therefore, in this protocol federate F1 will publish the "entity" parameter and subscribe to "Queuestatus" parameter of the interaction class, while federate F2 will subscribe to the "entity" parameter and publish the "Queuestatus" parameter of the interaction class. The expected value for the "Queuestatus" is Boolean, i.e., either the Queue is full or not full. Therefore, in this protocol "Queuestatus" is declared as an integer having only two values (i.e., 0: not full; and 1: Full). It is important to note that according to the OMT specification "HLAinteractionRoot" remains the superclass of all other interaction classes in Federate Object Model (FOM) and is mandatory. Also, the given protocol will use zero lookahead. The FOM for this protocol is defined as below.

```
<interactions>
    <interactionClass>
        <name>HLAinteractionRoot</name>
        <sharing>PublishSubscribe</sharing>
        <dimensions>NA</dimensions>
        <transportation>HLAreliable</transportation>
        <order>Receive</order>
        <interactionClass>
            <name>EntityTransfer</name>
            <sharing>PublishSubscribe</sharing>
            <transportation>HLAreliable</transportation>
            <order>TimeStamp</order>
            <parameter>
                <name>entity</name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name> Queuestatus </name>
                <dataType>HLAinteger32BE</dataType>
```

```
20.             </parameter>
21.          </interactionClass>
22.      </interactionClass>
23. </interactions>
```

In this scenario, Model M1 will produce an entity at time T that needs to be transferred to Model M2. Time to transfer is identified at run time, i.e., it is not fixed as discussed in Chapter 4. Each simulation model uses exponential distribution and all the models are based on stochastic approach. An entity arriving in Model M1 is generated using an exponential distribution where α=1.5. Similarly, the service time for Activity A1 in Model M1 and Activity A2 in Model M2 also uses the exponential distribution where α=1. However, there will be some different test runs with different distribution values to monitor changes in performance. Also, this scenario uses zero lookahead and it assumes that entity travel time from one model to another is zero. Figure 5.9 illustrates the sequence diagram (SD) of the following proposed approach based on the protocol explained above. Similarly, Figure 5.10 illustrate the flow of the protocol algorithm.

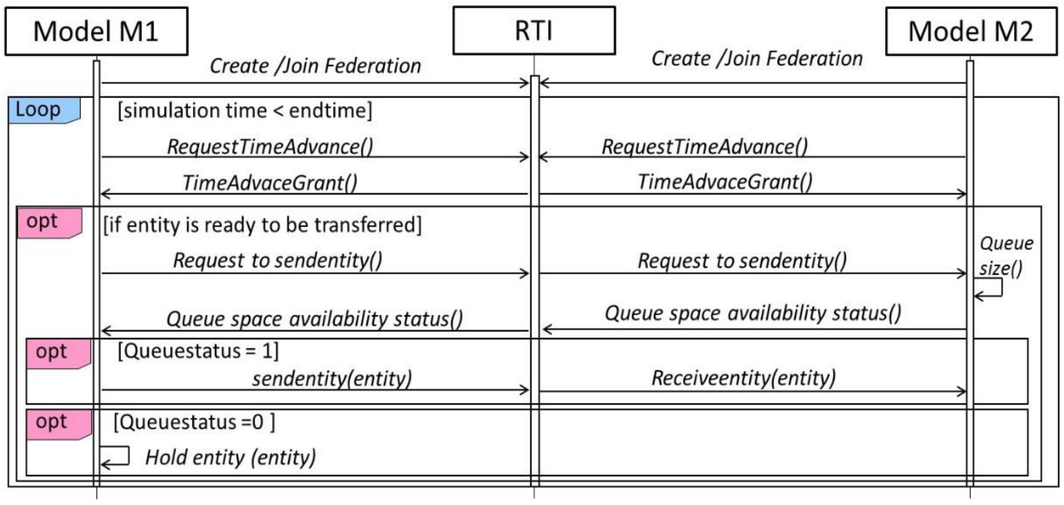


Figure 5.9: SD Queue Update Protocol using RTI

Sequence of Events

- **Initialisation stage:** Both federates will either create or join federation.

- **Initialisation stage:** Federate F1 will indicate that it is capable of sending entities by publishing "entity" interaction class parameter and by subscribing to "Queuestatus" interaction class parameter to any destination federate that is subscribing to "entity" and publishing "Queuestatus" interaction class parameter.
- **Initialisation stage:** Federate F2 will indicate that it is capable of receiving entities by subscribing to "entity" and publishing "Queuestatus" interaction class parameter for any source federate who is publishing "entity" and subscribing "Queuestatus" interaction class parameter.
- **Simulation Runtime:** Both Federate F1 and F2 will progress time "t" using the time management service of RTI.
- **Simulation Runtime:** At a particular time "T1", the element A1 in Federate F1 will have an entity to be transferred to Q2 in Federate F2. Therefore, Federate F1 will first send a request to Federate F2 via RTI Ambassador, to check space in Q2 and it will wait for the response. Federate F2 will check the space and send the status of Q2 using "Queuestatus" parameter via RTI Ambassador.
    - **Case 1:** If Q2 in Federate F2 can accommodate a new entity e1 from element A1 then "1" value message will be sent to Federate F2. A1 in F1, then will send an entity e1 to Q2 in F2 using Federate Ambassador and request time advance to RTI. Similarly, Federate F2 will update its queue and request time advance to RTI and simulate progress normally.
    - **Case 2 (a):** If Q2 in Federate F2 is full and cannot accommodate new entity e1 from A1 then a "0" value message will be sent to Federate F2, and F2 will request time advance to RTI. Element A1 in Federate F1 will hold the entity and will block the processing of the element A1 (only). Federate F1 then will request time advance from RTI.
    - **Case 2(b):** After a successful time advance, at time T2, element A1 from Federate F1 will attempt to send the entity e1 again (using the above defined

processes in Case 1 or 2(a)). If Q2 is not full at T2 the entity e1 will be transferred to Q2 in Federate F2 from the element A1 in Federate F1 (as defined in case 1) and both federates will request time advance. But, if Q2 is still full at T2 the element A1 in Federate F1 will remain blocked (as defined in case 2(a)) and both federates will request time advance.

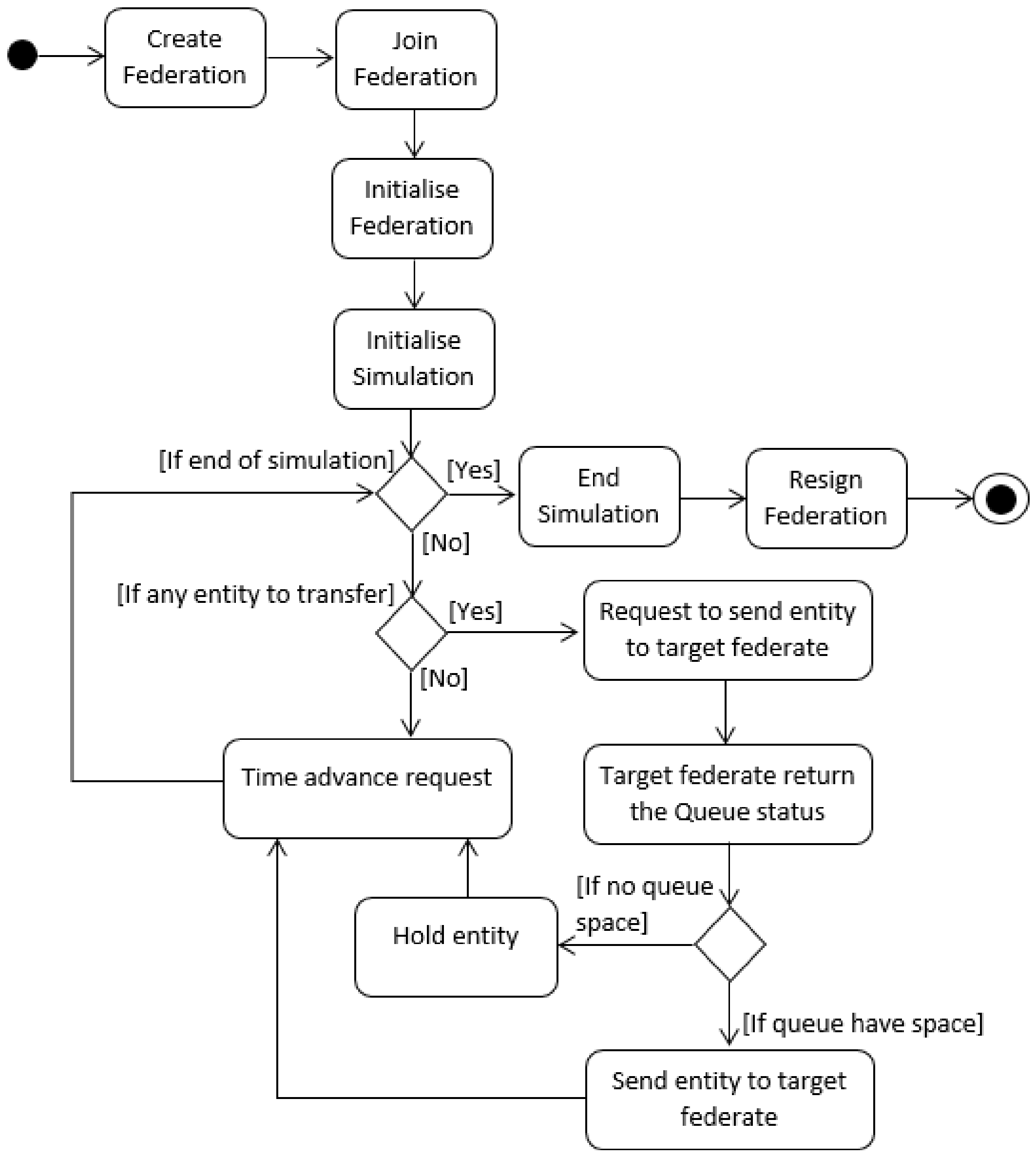


Figure 5.10: Activity Diagram for Queue Update Protocol

5.2.3.2 Bounded Receiving Element using Bounded Buffer Update Protocol

The second possible approach to address IRM Type A.2 bounded receiving element is by slightly modifying the previous protocol to reduce the message passing between federates by sending bounded buffer updates. The relationship between the Models, CSP, and RTI is shown in Figure 5.8 above. For the purpose of understanding the entity transfer concept for IRM Type A.2 Bounded Receiving Element, it is necessary to identify the source and destination model. Therefore, similar to the last protocol, it will be assumed that Federate F1 is a source, M1 is the model from which an entity leaves, and Federate F2 implements a destination model M2 at which the entity arrives. The queue Q2 in model M2 is declared as a Bounded Receiving element. Time is defined as the time when an entity exits a model and may immediately arrive at the destination model, i.e., an event happens in M1 at time “T1” marking the departure of an entity from M1 and instantly arriving at M2 at “T2” (where T1=T2=t).

The message passing in this protocol is represented as a mixture of the Object class and Interaction class with one parameter for “entity” and another attribute for “Queuestatus” in both the models. As there are only two models in a given scenario and the source and destinations are defined, we do not need to define the destination and source names. This protocol will also use zero lookahead. The FOM for this protocol is defined as below.

```
1.  <objects>
2.      <objectClass>
3.      <name>HLAobjectRoot</name>
4.      <sharing>Neither </sharing>
5.      <objectClass>
6.          <name> EntityTransfer</name>
7.          <sharing>PublishSubscribe</sharing>
8.          <attribute>
9.              <name> Queuestatus </name>
10.             <dataType>HLAinteger32BE </dataType>
```

```
            <updateType>Conditional </updateType>
            <ownership>NoTransfer</ownership>
            <sharing>PublishSubscribe</sharing>
            <dimensions>NA </dimensions>
            <transportation>HLAreliable</transportation>
            <order>TimeStamp</order>
        </attribute>
    </objectClass>
</objects>
<interactions>
    <interactionClass>
        <name>HLAinteractionRoot</name>
        <sharing>PublishSubscribe</sharing>
        <dimensions>NA</dimensions>
        <transportation>HLAreliable</transportation>
        <order>Receive</order>
        <interactionClass>
            <name>EntityTransfer</name>
            <sharing>PublishSubscribe</sharing>
            <transportation>HLAreliable</transportation>
            <order>TimeStamp</order>
            <parameter>
                <name>entity</name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
        </interactionClass>
    </interactionClass>
</interactions>
```

In this scenario, model M1 will produce an entity at time T that needs to be transferred to model M2. Time to transfer is identified at run time. Each simulation model uses exponential distribution and all the models are based on a stochastic approach. The entity arriving in model M1 is generated using an exponential distribution where α=1.5. Similarly, the service time for activity A1 in model M1 and activity A2 in model M2 also uses exponential distribution where α=1. However, there will be some different test runs with different distribution values to monitor the change in performance.

Federate F1 will publish the “*entity*” parameter of the interaction class and subscribe to the “*Queuestatus*” attribute of the object class, while Federate F2 will subscribe to the “entity” parameter from the interaction class and publish the “Queuestatus” attribute of the Object class. The expected value for the “Queue status” is the size of the queue. Therefore, “Queuestatus” is declared as an integer having only values (i.e., value greater than 1: representing Queue not full and 0: representing Queue Full). Federate F1 will provide the instance parameter values of entity, hence it will be *updating* the entity instance parameter while Federate F2 will receive the instance parameter values of entity and will be *reflecting* the entity instance parameter.

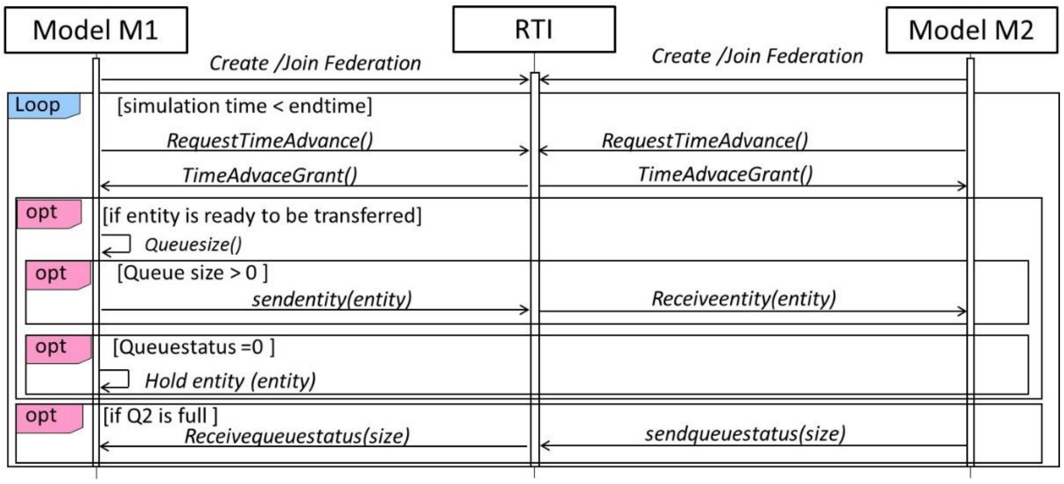


Figure 5.11: SD Bounded Buffer Update Protocol using RTI

Similarly, Federate F2 will provide the Object attribute values of Queuestatus, hence it will be *updating* Queuestatus Object attribute while Federate F1 will receive the Object attribute value of Queuestatus, hence it will be *reflecting* the Queuestatus Object attribute. Model M1 will continue to send an entity until it receives a full queue message from model M2 via Queuestatus. The entity will be held at A1 in model M1 until the other federate does not send a message for available space in the bounded queue in model M2 via Queuestatus. The model M2 will indicate to sender when there is no space left in the queue, this will allow model M1 to stop activity A1 initiating further events after completing the current event. Hence, in this protocol model M1 will not request the queue status from model M2. While Model M2 will update the status to Model M1 once the Queue is full. Figure 5.11 illustrates the sequence diagram (SD) of the following proposed approach based on the protocol defined above. Also Figure 5.12 illustrate the flow of the protocol algorithm.

Sequence of Events

- **Initialisation stage:** Both Federates will either create or join federation**.**
- **Initialisation stage:** Federate F1 will indicate that it is capable of sending entities by publishing "entity" interaction class parameter and by subscribing to "Queuestatus" Object class attribute for any destination federate that is subscribing to "entity" and publishing the "Queuesize" parameter and attribute.
- **Initialisation stage:** Federate F2 will indicate that it is capable of receiving entities by subscribing to "entity" interaction class parameter and publishing "Queuestatus" Object class attribute for any source federate who is publishing "entity" and subscribing "Queuesize" parameter and attribute.
- **Simulation Runtime:** Federate F2 will publish the Queuestatus in Q2 via RTI Ambassador and indicate if the Queue Q2 is full or has space to accommodate more entities. Both Federates F1 and F2 will progress time "t" using the time management service of RTI.
- **Simulation Runtime:** At a particular time "T1", the Element A1 in Federate F1 will have an entity to be transferred to Q2 in Federate F2. Federate F1 will check against the local copy of Queuestatus (previously updated by Federate F2 at Time < T1). Similarly, on every change of the state of the queue, Federate F2 will update the Queuestatus attribute via RTI ambassador.

- **Case A:** If the value of Queuestatus is “1” or greater, this indicates that the Queue Q2 in Federate F2 is not full and can accommodate new entity e1 from A1. Entity e1 will then be transferred from Federate F1 via RTI ambassador and both Federate F1 and F2 will progress time “t” using the time management service of RTI.
- **Case B:** If the value of Queuestatus is “0” this indicates that the Queue Q2 in Federate F2 is full and cannot receive any entities; therefore, the element A1 will hold the entity and ***block*** the processing of the element A1 (only) and both Federates F1 and F2 will progress time “t” using the time management service of RTI. While element A1 is blocked, on every successive time advance Federate F1 will check the value of Queuestatus and at any time the value is updated to “0” by Federate F2 attribute update, the Case A scenario will be executed to successfully transfer the entity.

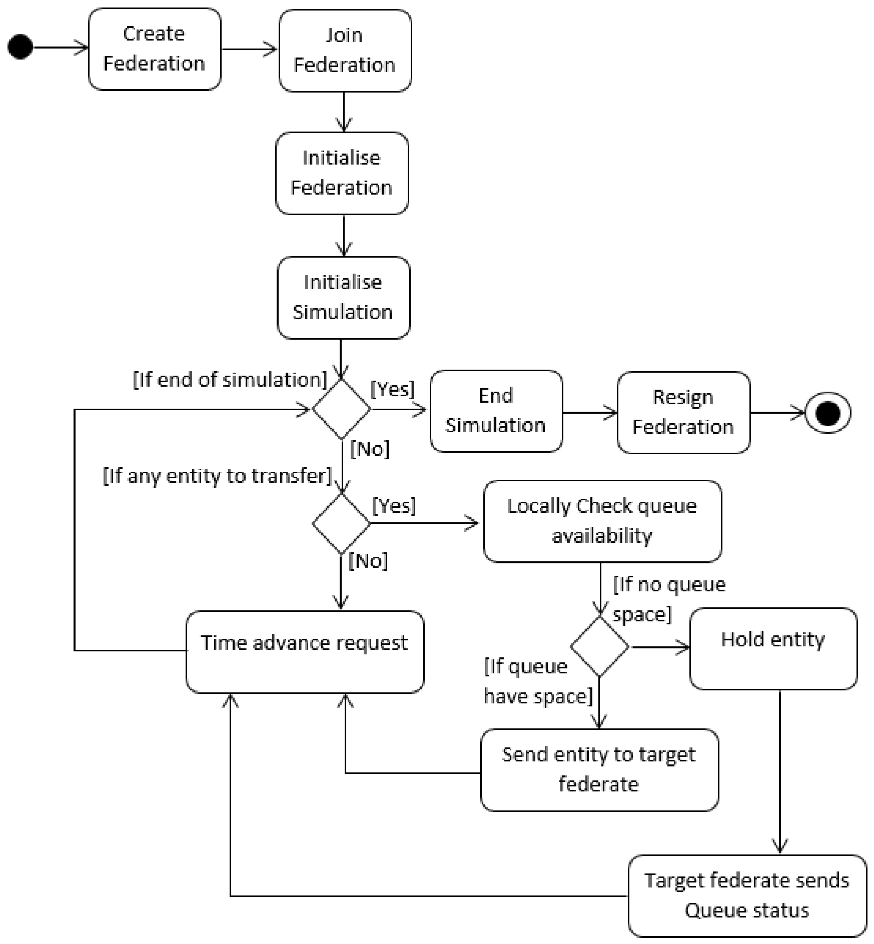


Figure 5.12: Activity diagram Bounded Buffer Update

5.2.3.3 Bounded Receiving Element using Adaptive Protocol

The third possible approach to address IRM Type A.2 bounded receiving element is by using an adaptive approach. In this protocol The message passing is limited and done when necessarily required. The relationship between the Models, CSP and RTI is as shown in Figure 5.8 above. This protocol will have the same assumption about the models, i.e., Federate F1 is a source model from which an entity leaves and Federate F2 is the destination model at which the entity arrives and contains a bounded queue Q2. Time will be defined as the time when an entity exits a model and may immediately arrive at the destination model, i.e., an event happens in M1 at time "T1" marking the departure of an entity from M1 and instantly arriving at M2 at "T2" (where T1=T2=t).

Message passing in this protocol is represented by a single interaction class with two parameters "entity" and "Queuesize" in both the models. Further parameters or attributes could be adjusted based on the simulation requirements and the nature of the entity. Federate F1 will publish the "entity" parameter and subscribe to the "Queuesize" parameter of the interaction class, while Federate F2 will subscribe to the "entity" parameter and publish the "Queuesize" parameter of the interaction class. The expected value for the "Queuesize" is a positive integer value, unlike the previous implementation, i.e., available Queue slots to receive entities. Therefore, "Queuesize" is declared as Integer where Queuesize => 0. Also, in Federate F1 and Federate F2 a local queue size variable "QSize" will be created and initialised to Zero. It is important to note that according to the OMT specification "HLAinteractionRoot" remains the superclass of all other interaction classes and is mandatory. Also, the given example will use zero lookahead. The FOM for this protocol is defined as the following:

```
1. <interactions>
2.     <interactionClass>
3.         <name>HLAinteractionRoot</name>
4.         <sharing>PublishSubscribe</sharing>
5.         <dimensions>NA</dimensions>
```

```
        <transportation>HLAreliable</transportation>
        <order>Receive</order>
        <interactionClass>
            <name>EntityTransfer</name>
            <sharing>PublishSubscribe</sharing>
            <transportation>HLAreliable</transportation>
            <order>TimeStamp</order>
            <parameter>
                <name>entity</name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name> Queuesize </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
        </interactionClass>
    </interactionClass>
</interactions>
```

Both models will maintain “QSize” local variable with the max size of the Queue (in this case the initial size is set to 10). This size will reduce by one on each successful entity transfer from Model M1 to Model M2. The entities will transfer until the QSize is reduced to zero, at this stage Model M1 will expect a new available QSize from Model M2. This new size will be once again updated to “QSize” local variable. This process will continue until the end of the simulation as illustrated in Figure 5.13. In this scenario, Model M1 will produce an entity at time T that needs to be transferred to Model M2. Time to transfer is identified at run time. Each simulation model uses exponential distribution and all the models are based on a stochastic approach. Entity arriving in Model M1 is generated using an exponential distribution where α=1.5. Similarly, the service time for Activity A1 in Model M1 and Activity A2 in Model M2 also uses exponential distribution where α=1. However, there will be some different test runs with different distribution values to monitor the change in performance. Figure 5.14 illustrate the flow of the protocol algorithm.

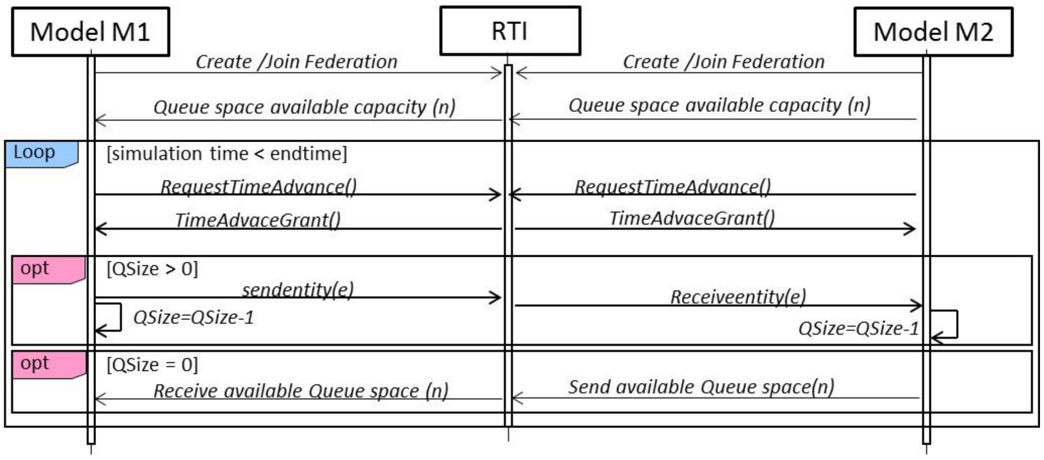


Figure 5.13: SD Adaptive Protocol using RTI

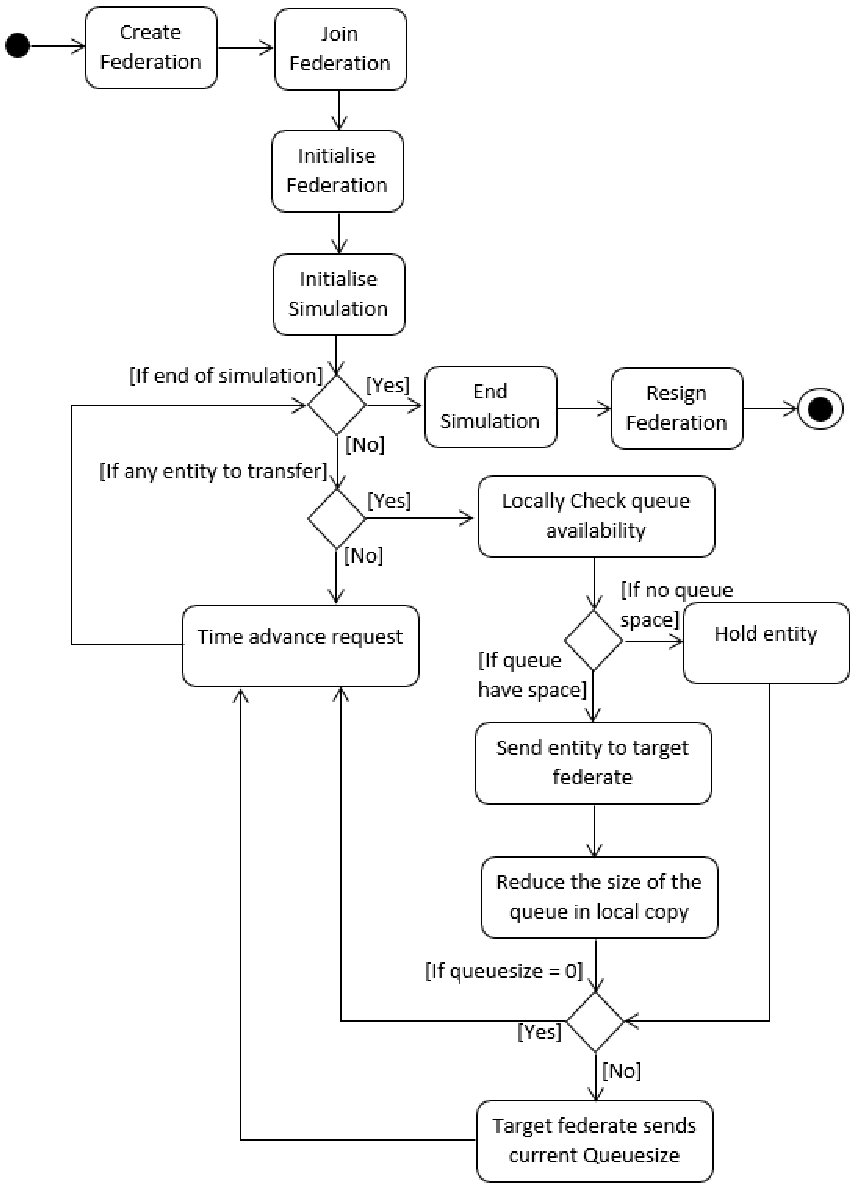


Figure 5.14: Activity Diagram using Adaptive Protocol

Sequence of Events

- **Initialisation stage:** Both Federates will either create or join a federation**.**
- **Initialisation stage:** Federate F1 will indicate that it is capable of sending entities by publishing "entity" interaction class parameter and by subscribing to "Queuesize" interaction class parameter from any destination federate that is subscribing to "entity" and publishing "Queuesize" interaction class parameter.
- **Initialisation stage:** Federate F2 will indicate that it is capable of receiving entities by subscribing to "entity" and publishing "Queuesize" interaction class parameter for any source federate that is publishing "entity" and subscribing to "Queuesize" interaction class parameter.
- **Simulation Runtime:** Federate F2 will publish the available space in Q2 via RTI Ambassador and also update its local QSize variable. Also, Federate F1 will record the available space in its QSize local variable. Both Federates F1 and F2 will progress time "t" using the time management service of RTI.
- **Simulation Runtime:** At a particular time "T1", the Element A1 in Federate F1 will have an entity to be transferred to Q2 in Federate F2. Therefore, Federate F1 will check its local variable QSize to see if Q2 has space. On each successful transfer of entity this variable is subtracted by "1", i.e., unlike the previous solution Federate F1 will not request the queue size, instead it will wait for the other federate to update the queue size. Therefore, at any time in simulation if QSize becomes Zero "0", Federate F2 will update its local variable QSize (with the current available space in Queue) and publish the available space via RTI Ambassador. Federate F1 will also update its local QSize variable with the new value.
  - **Case A:** If Q2 in Federate F2 can accommodate a new entity e1 from Element A1, i.e., the QSize > 0, then A1 in F1 will send an entity e1 to Q2 in F2 using federate Ambassador. Both federates at this point will subtract the number of entities from QSize (i.e., in this case QSize =

QSize-1) and request time advance to RTI and simulation progress normally.

- **Case B:** If QSize becomes zero "0" in Federate F1, then Federate F1 will block the processing of the Element A1 (only) and request time advance. Similarly, Federate F2 will also request time advance and the simulation will progress normally. Once Q2 has space in time Tn to accommodate more entities, Federate F2 will update its local QSize variable and publish the available space via RTI Ambassador. Federate F1 will update its local variable QSize and proceed to case A. Element A1 will be released to continue its processing with the next entity in line.

5.2.4 Multiple Input Prioritization

IRM Type A.3 Multiple Input Prioritization represents an interoperability problem that occurs when a model element can receive entities from more than one different source. The receiving element in a model could be a Queue. Note that this IRM does not include cases where the receiving element is bounded as described in IRM Type A.2. This problem is already defined in Chapter 4 in more detail; therefore, this section will focus on the conceptual modelling of each protocol used to address this IRM issue. There are two protocols identified to address this problem.

5.2.4.1 Multiple Input Specialised Prioritisation Protocol

The first possible approach to address IRM Type A.3 multiple input prioritisation is by using a specialised prioritisation of receiving entities. The relationship between the Models, CSP and RTI is shown in Figure 5.15. In the multiple input prioritisation protocol, it is necessary to identify the source and destination model(s). Therefore, it will be assumed that Federate F1 and Federate F2 are source models from which an entity leaves and Federate F3 a destination model at which the entity(s) arrive either simultaneously or at different times. Time will be defined as the time when an entity exits a model and may immediately arrive at the destination model, i.e., an event happens in M1 at time "T1" marking the departure of an entity from M1 and instantly arriving at M3 at "T2" (where T1=T2=t). For the purpose of explaining Multiple Input Prioritisation requirements, it will be assumed that the entities arrive at Federate F3 at the same time from Federate F1 and Federate F2.

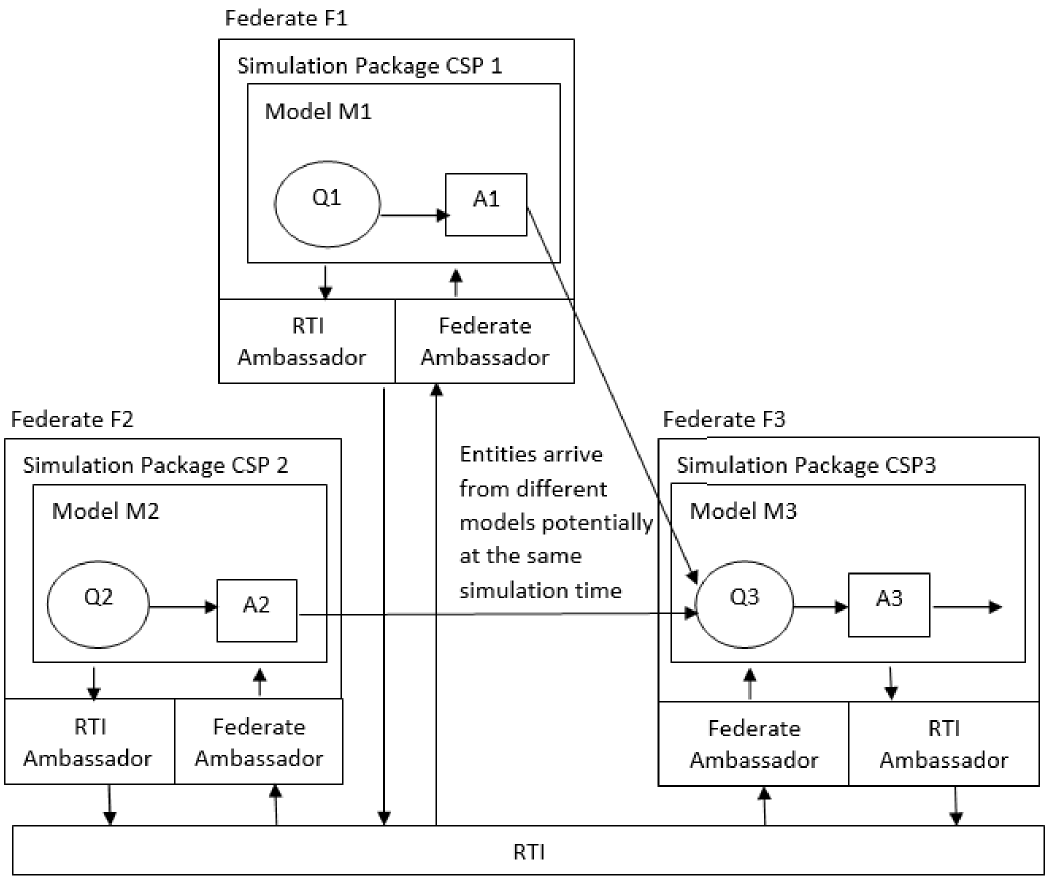


Figure 5.15: Multiple Input Prioritisation using HLA

To transfer an entity from Model M1 and Model M2 in Federate F1 and Federate F2 to model M3 in Federate F3, the message passing is represented by an interaction class with number of attributes defined by the type of entity. Again for the purpose of understanding, entity in this protocol is represented as a single interaction class with two parameters "entity" and "ModelID" in all source models. Model ID parameter is used because there are more than two models in a given scenario therefore it will ease the identification of the source model. Since each of the source and destination federates are defined separately by RTI, the information related to source identity could be extracted by the destination federate, but for simplicity it is included as an identifier. Alternatively, the entity could also be defined as an object, having an identifier as an attribute value of the object.

The following is the FOM used to define the interaction class.

```
<interactions>
    <interactionClass>
        <name>HLAinteractionRoot</name>
        <sharing>PublishSubscribe</sharing>
        <dimensions>NA</dimensions>
        <transportation>HLAreliable</transportation>
        <order>Receive</order>
        <interactionClass>
            <name>EntityTransfer</name>
            <sharing>PublishSubscribe</sharing>
            <transportation>HLAreliable</transportation>
            <order>TimeStamp</order>
            <parameter>
                <name>entityF1</name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name> ModelIDF1  </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name>entityF2</name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name> ModelIDF2</name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
        </interactionClass>
    </interactionClass>
</interactions>
```

Using the above given FOM, Federate F1 and Federate F2 will publish the "entityFx" ("x" represents the federate identifier number) and "ModelIDFx" parameter of the interaction class, while the SOM for Federate F3 will subscribe to the "entityFx" and "ModelIDFx" of the interaction class. In this case of specialised prioritisation, a higher priority for Federate F1 over Federate F2 entities has been set in Federate F3, i.e., if Federate F3 receives entities from Federate F1 and Federate F2 at the same time, and the Queue Q3 of Federate F3 implements the FIFO strategy, then the entity from Federate F1 will be added to Queue Q3 before inserting the entity from Federate F2. This protocol will also use zero lookahead.

In this protocol each simulation model uses normal distribution and all the models are based on a stochastic approach. Both simulation models, i.e., Model M1 and Model M2 use the same normal distribution for generating the new entities (i.e. Arrivals) where the mean = 4.0 and Standard deviation SD = 1.5 (i.e. μ=4, σ = 1.5). The service time for Activity A1, Activity A2 and Activity A3 are also defined by Normal Distribution of mean = 4.0 and Standard deviation SD = 1.5 (i.e. μ=4, σ = 1.5). Figure 5.16 illustrates the sequence diagram of the proposed approach based on the protocol explained above and Figure 5.17 illustrate the flow of the protocol algorithm.

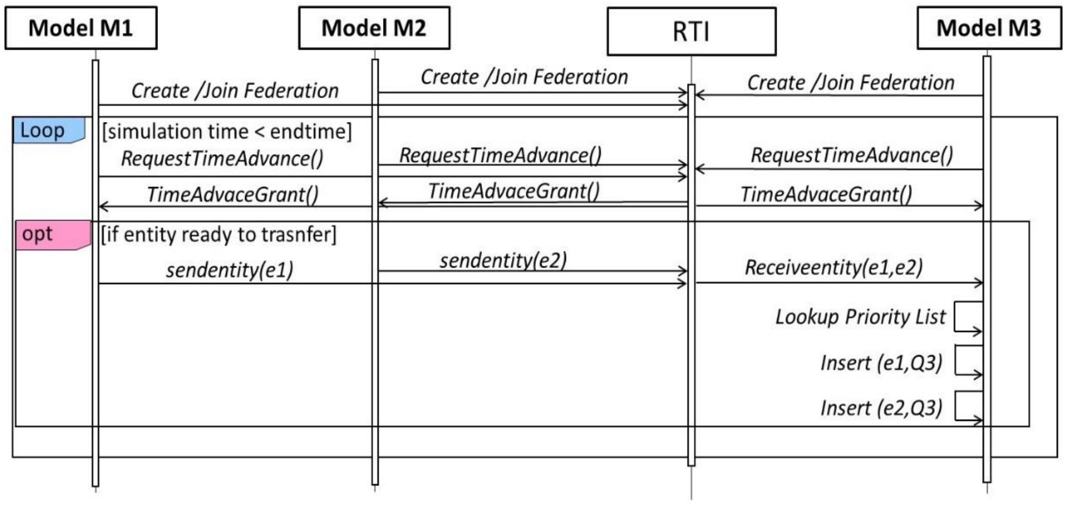


Figure 5.16: SD Multiple Input Specialised Prioritisation using RTI

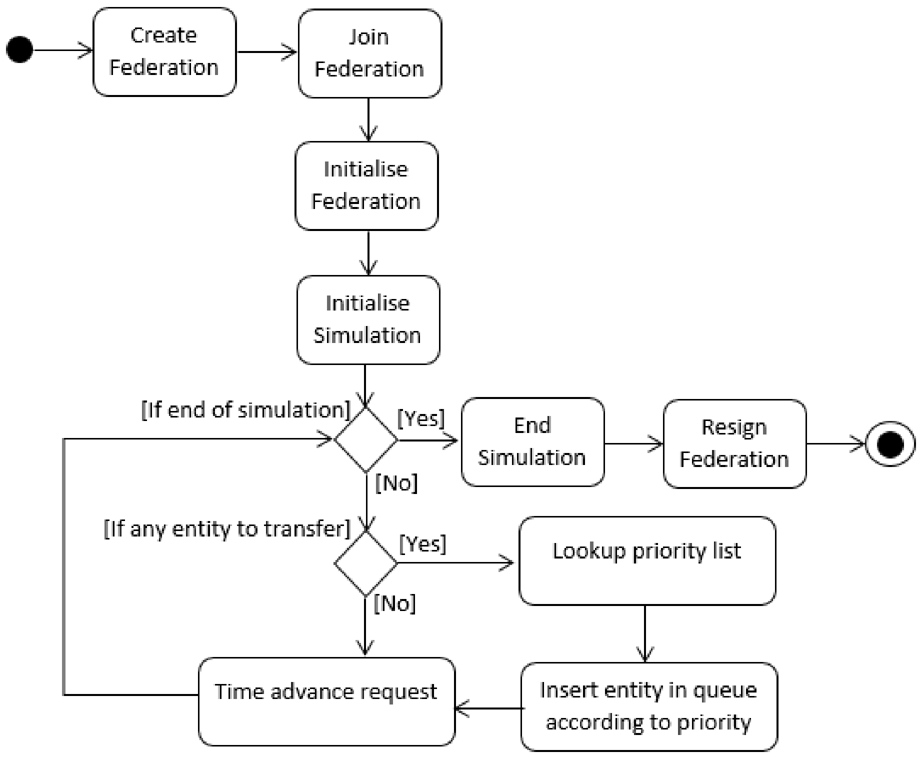


Figure 5.17: Activity Diagram for Multiple Input Specialised Prioritisation

Sequence of Events

- **Initialisation stage:** Both Federate will either create or join a federation.
- **Initialisation stage:** Federate F1 and Federate F2 will indicate that they are capable of sending entities by publishing "entity" and "ModelID" interaction class parameter to any destination federate who is subscribing to "entity" and "ModelID" interaction class parameter.
- **Initialisation stage:** Federate F3 will indicate that it is capable of receiving entities by subscribing to "entity" and "ModelID" interaction class parameter for any source federate who is publishing "entity" and "ModelID" interaction class parameter.
- **Simulation Runtime:** Both Federates F1 and F2 will progress time "t" using the time management service of RTI.
- **Simulation Runtime:** At a particular Time "T1" the element A1 in Federate F1 will have an entity to be transferred to Queue Q3 in Federate F3.

Similarly, at the same time "T1" the element A2 in Federate F2 will also have an entity to be transferred to Queue Q3 in Federate F3. Both the entities will arrive at time "T1" in Queue Q3 of Federate F3.

- **Simulation Runtime:** Federate F3 will check against its specific list for priority against the ModelID received from both federates. As illustrated above, the specific list was set to have higher priority for Federate F1 over Federate F2. Therefore, the entity from Federate F1 will be inserted into Queue Q3 before inserting the entity received from Federate F2 (based on the predefined priority order in the model).

5.2.4.2 Multiple Input Dynamic Prioritisation Protocol

The second possible approach to address IRM Type A.3 multiple input prioritisation is by using a dynamic prioritisation of receiving entities. The relationship between the Models, CSP and RTI is shown in Figure 5.15 above. The source and destination model(s) are the same as defined in the previous protocol. Therefore, Federate F1 and Federate F2 are source models from which an entity leaves and Federate F3 a destination model at which the entity(s) arrives either simultaneously or at different times. Time T will be defined as the time when an entity exits a model and might immediately arrive at the destination model. Similarly, in this protocol it will be assumed that the entities arrive at Federate F3 at the same time from Federate F1 and Federate F2.

In relation to transferring an entity from Model M1 and Model M2 in Federate F1 and Federate F2 to Model M3 in Federate F3, it is represented as a single interaction class with two parameters "entity" and "ModelID" in all source models as described in the previous section. The FOM used for this protocol is the same as the one used for Multiple Input Specialised Prioritisation. Therefore, Federate F1 and Federate F2 will publish the "entityFx" and "ModelIDFx" parameters of the interaction class, while Federate F3 will subscribe to all "entityFx" and "ModelIDFx" of the interaction class. In this case of dynamic prioritisation, Federate F3 will decide the priority for Federate F1 and Federate F2 entities at run time, i.e., if Federate F3 receives two entities from Federate F1 and Federate F2 at the same time and the

Queue Q3 of Federate F3 implements the FIFO strategy, then Model M3 will make a decision based on the simulation requirement. Once the priority list is determined, the entity from a federate with the highest priority will be added to Queue Q3 followed by the entity from the other federations. This priority list could stay fixed (once determined at the first transfer of entity) for the duration of the simulation or could change every time Federate F3 receives entities. But for this protocol the model will determine the priority list every time when it receives the simultaneous entity.

In this protocol each simulation model uses normal distribution and all the models are based on a stochastic approach. Both simulation models, i.e., Model M1 and Model M2 use the same normal distribution for generating the new entities (i.e., Arrivals) where the mean = 4.0 and Standard deviation SD = 1.5 (i.e. μ=4, σ = 1.5). The service time for Activity A1, Activity A2, and Activity A3 are also defined by Normal Distribution of mean = 4.0 and Standard deviation SD = 1.5 (i.e. μ=4, σ = 1.5). Figure 5.18 illustrates the sequence diagram of the proposed approach based on the protocol explained above and Figure 5.19 illustrate the flow of the protocol algorithm.

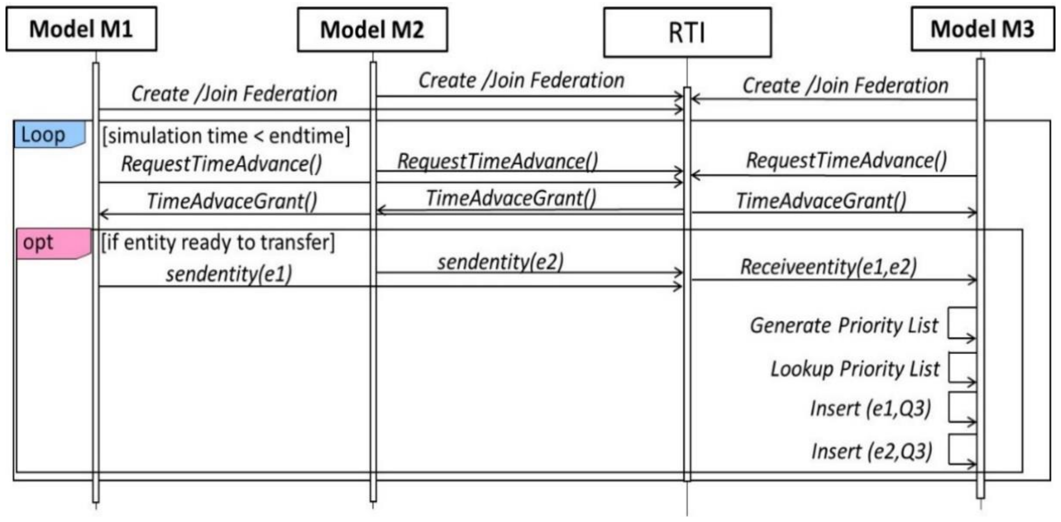


Figure 5.18: SD Multiple Input Dynamic Prioritisation

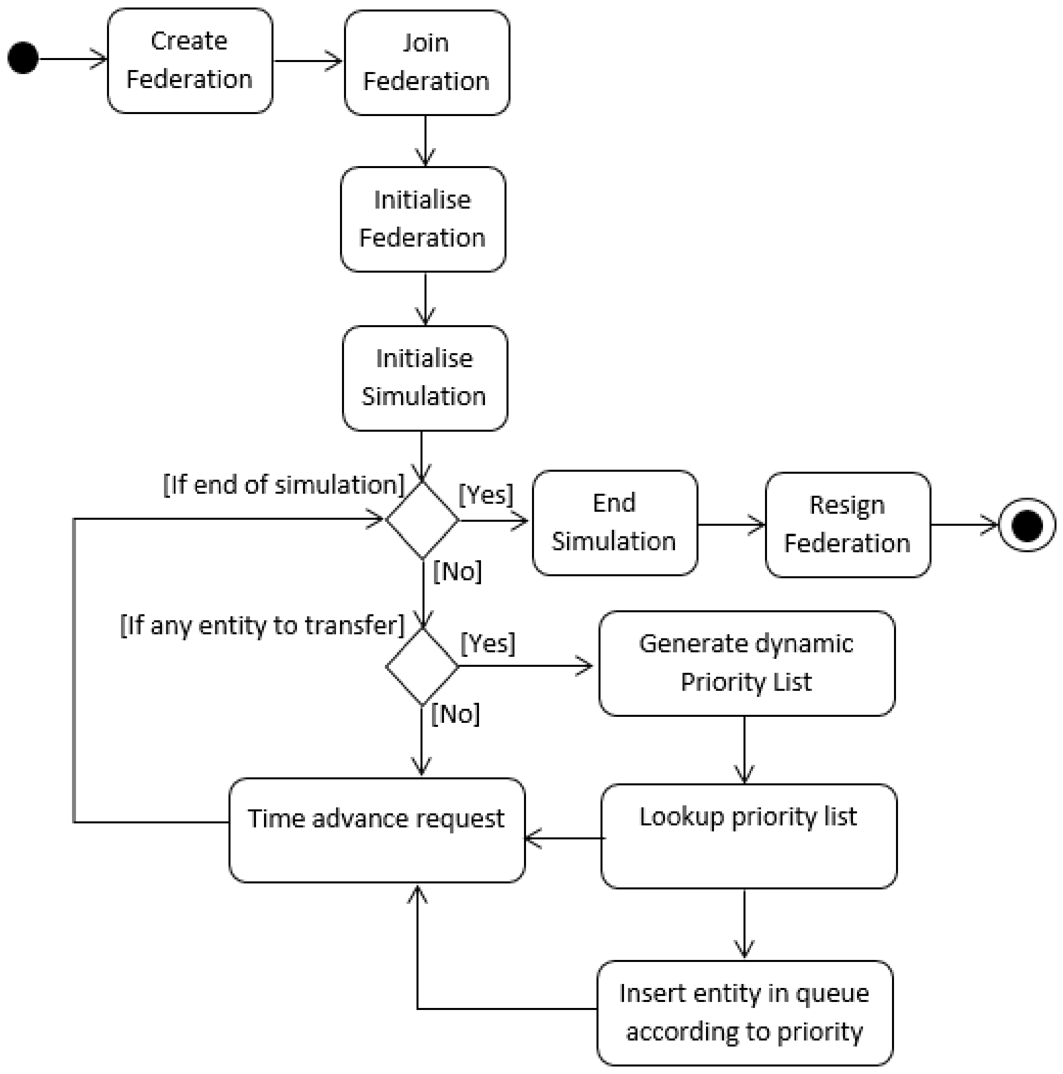


Figure 5.19: Activity Diagram for Multiple Input Dynamic Prioritisation

Sequence of Events

- **Initialisation stage:** Both Federates will either create or join federation.
- **Initialisation stage:** Federate F1 and Federate F2 will indicate that they are capable of sending entities by publishing "entityFx" and "ModelIDFx" interaction class parameter to any destination federate that is subscribing to "entityFx" and "ModelIDFx" interaction class parameter.
- **Initialisation stage:** Federate F3 will indicate that it is capable of receiving entities by subscribing to "entity" and "ModelID" interaction class

parameter for any source federate that is publishing "entity" and "ModelID" interaction class parameter.

- **Simulation Runtime:** Both Federate F1 and F2 will progress time "t" using the time management service of RTI.
- **Simulation Runtime:** At a particular Time "T1" the Element A1 in Federate F1 will have an entity to be transferred to Queue Q3 in Federate F3. At the same time "T1", the Element A2 in Federate F2 will also have an entity to be transferred to Queue Q3 in Federate F3. Both the entities will arrive at time "T2" in Queue Q3 of Federate F3 (where T1=T2, i.e., zero travel time).
- **Simulation Runtime:** The Federate F3 will generate the priority list and then select entities priority against its new priority list from both Federate F1 and Federate F2. The entities will be inserted into the Queue Q3 in Federate F3 from Federate F1 and Federate F2 accordingly.

5.2.5 General Shared Resource

IRM Type B Shared Resource represents an interoperability problem that occurs when elements of two or more models share the same resource. The issue of sharing a resource is not only limited to concurrent requests, but also defines the strategy of maintaining resource consistency when shared between models running separately. This problem is already defined in Chapter 4 in more detail; therefore, this section will focus on the conceptual modelling of each protocol used to address this IRM issue. There are four different protocols identified to address this problem. The relationship between the Models, CSP and RTI is shown in Figure 5.20 below.

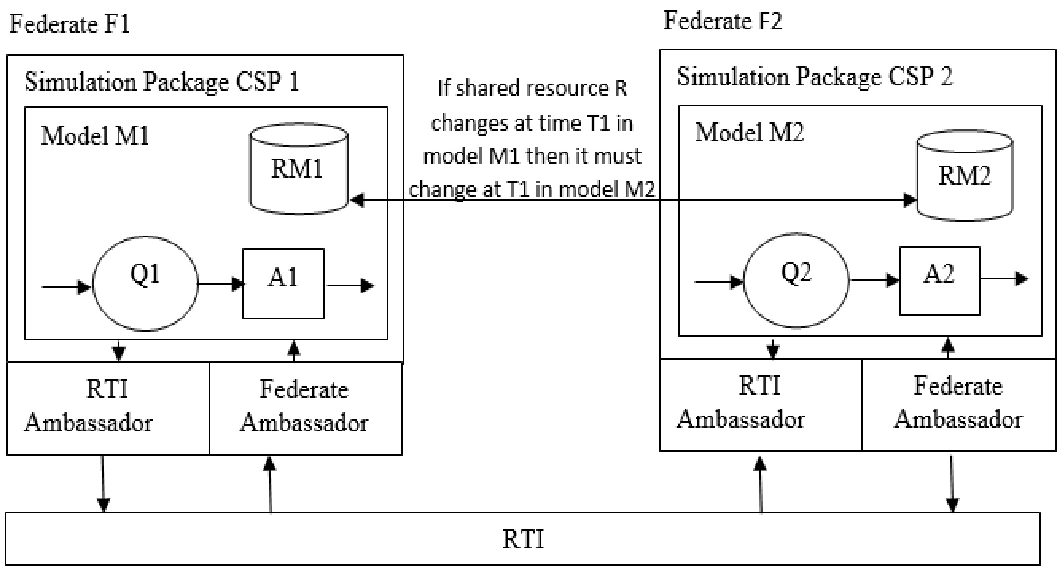


Figure 5.20: General Shared Resource using HLA

5.2.5.1 Continuous Resource Update Protocol

The first possible approach to address IRM Type B.1 general shared resource is by using a continuous resource update approach. The relationship between the Models, CSP and RTI is shown in Figure 5.20 above. For the purpose of understanding the general shared resource, it is necessary to identify the copies of Resource R and the models sharing the resource. A shared resource is not typically owned by any federate or model, their presence in a model is technical and logical. Therefore, a copy of a same resource is kept at each model as RMx. To maintain consistency among all copies of a shared resource at any given time T1 the models will keep these copies updated using message passing via RTI. Although a resource could be shared among more than two models, this example uses two federates, i.e., F1 and F2 for ease of understanding. Similarly, there could be more than one type of resource shared among federates for different purposes.

This protocol may represent a scenario where two units in a factory need attention for either fixed or variable time intervals and they need a machine engineer/ operator to acknowledge them. Further, this could also simulate the behaviour of a self-checkout till operator in a superstore, where an operator is assigned to multiple tills. This operator will intervene if any customer needs assistance or if the tills need approval for certain items.

To share the state of the resource among all copies of federates a resource could be represented as an object defining different attributes of the resource, e.g., a resource could perform more than one role, i.e., a machine operator or machine engineer, etc. Or there could be two different resources shared between the same models. The actual representation of the resource depends on the model implemented in a CSP and how the modeller will use the resource. Again for the purpose of understanding, resource in this protocol is represented as a single interaction class with one boolean parameter "requestFx" for each federate ("x" represents the federate number). The state of the request parameter could be either zero "0" indicating no resource request or one "1" for resource requests. Additionally, each federate will maintain a boolean variable in their local copies of a resource, i.e., RM1 and RM2. This variable will be used to record the status of the Resource at any given time, which will be consistent across the federation. The following FOM is used for this protocol.

```
<interactions>
    <interactionClass>
        <name>HLAinteractionRoot</name>
        <sharing>PublishSubscribe</sharing>
        <dimensions>NA</dimensions>
        <transportation>HLAreliable</transportation>
        <order>Receive</order>
        <interactionClass>
            <name> SharedResource </name>
            <sharing>PublishSubscribe</sharing>
            <transportation>HLAreliable</transportation>
            <order>TimeStamp</order>
            <parameter>
                <name> requestF1 </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name> requestF2 </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
        </interactionClass>
    </interactionClass>
</interactions>
```

Using the above given FOM, Federate F1 will publish the “requestF1” parameter and subscribe to “requestF2” parameter of the interaction class, while Federate F2 will publish the “requestF2” parameter and subscribe to “requestF1” parameter of the interaction class. Each federate will use a fixed / static priority table in the case of both model requests for the same resource occurring at the same time T. In this protocol, Federate F1 has higher priority for using the resource over Federate F2. Again, this priority table could also be dynamic or with a different strategy based on the model scenario and requirements.

In this protocol each simulation model uses normal distribution and all the models are based on a stochastic approach. Both simulation models, i.e., Model M1 and Model M2 use the same normal distribution for generating new entities (i.e. Arrivals) where the mean = 5.0 and Standard deviation SD = 1.7 (i.e., μ=5, σ = 1.7). The service time for Activity A1 and Activity A2 are also defined by Normal Distribution of mean = 4.0 and Standard deviation SD = 1.5 (i.e., μ=4, σ = 1.5). Figure 5.21 illustrates the sequence diagram of the proposed approach based on the protocol explained above. Figure 5.22 illustrate the flow of the protocol algorithm.

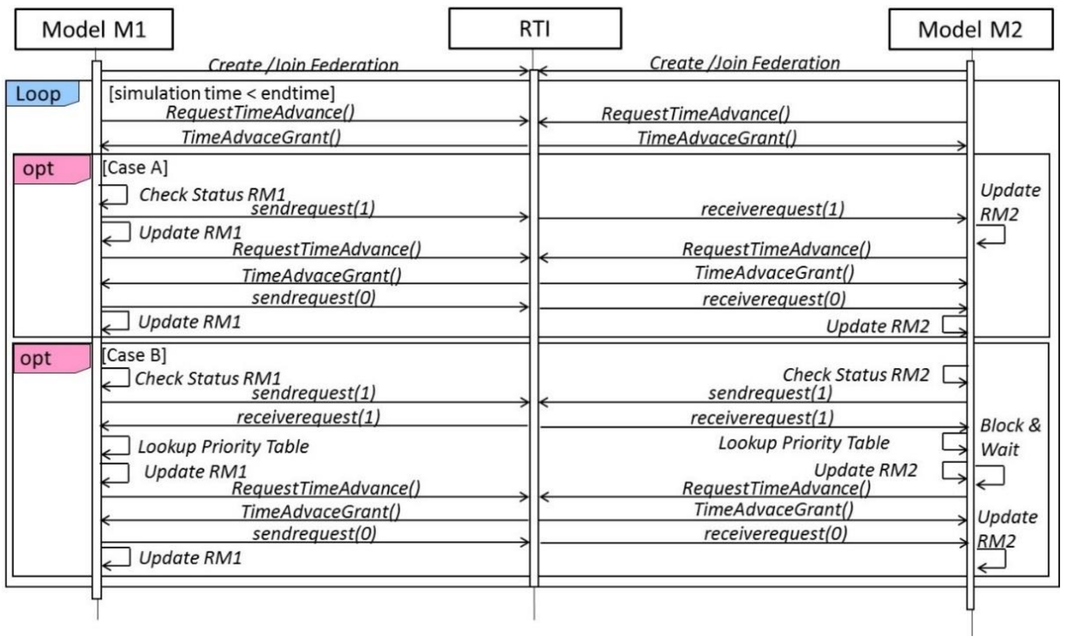


Figure 5.21: SD of General Shared Resource for Resource Update Protocol

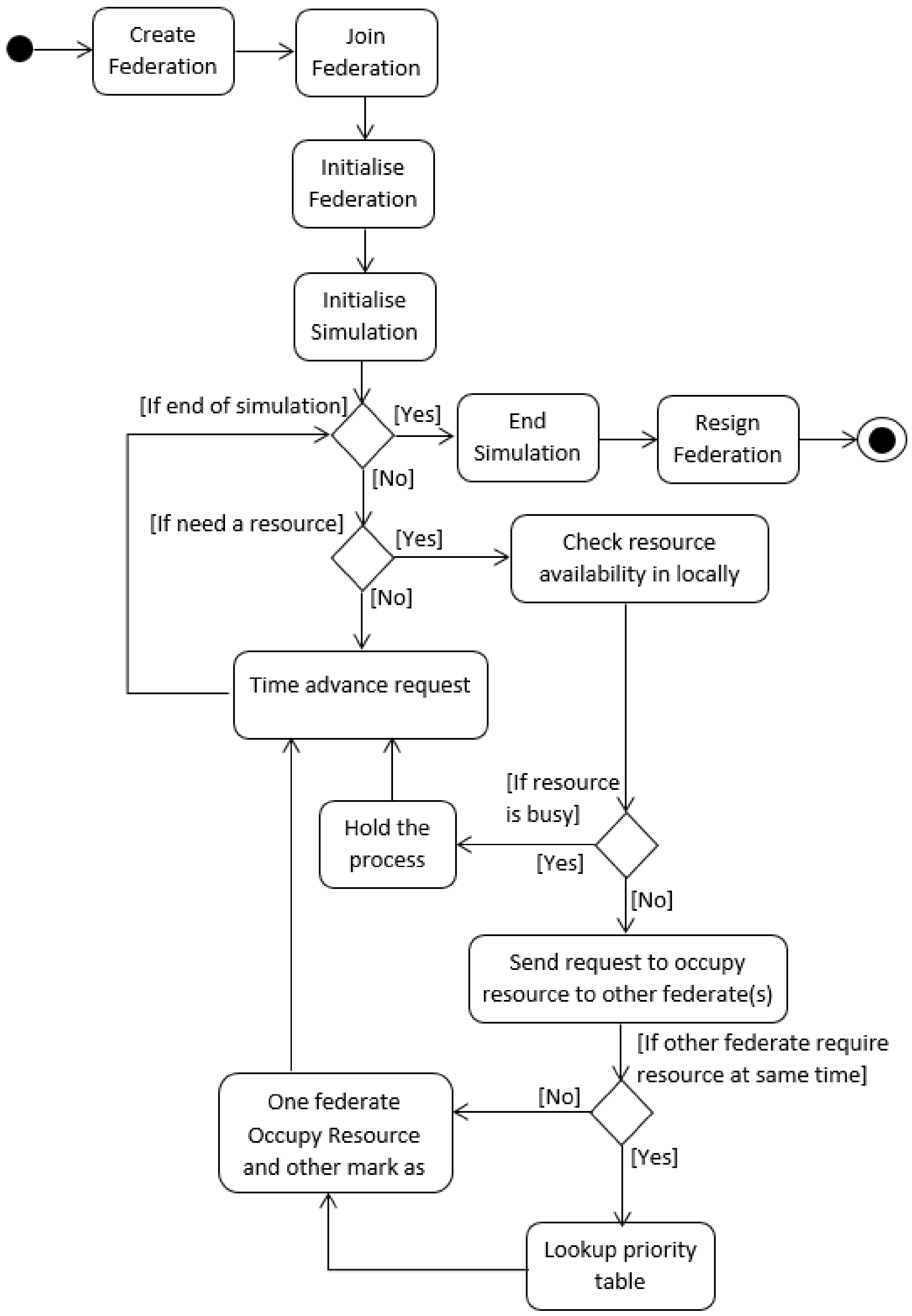


Figure 5.22: Activity Diagram of Shared Resource for Resource Update

Sequence of Events

- **Initialisation stage:** Both federates will either create or join federation.
- **Initialisation stage:** Federate F1 and Federate F2 will indicate that they are capable of sending resource requests by publishing "requestF1" and "requestF2" interaction class parameter to any destination federate that is subscribing to them.
- **Initialisation stage:** Federate F1 and Federate F2 will indicate that they are capable of receiving requests by subscribing to "requestF2" and "requestF1" interaction class parameter from any destination federate that is publishing them.
- **Simulation Runtime:** Both Federate F1 and F2 will progress time "t" using the time management service of RTI.
- **Simulation runtime Case A:**
  a) If Activity A1 in Federate F1 requires a resource at time T1 it will first check the status of the resource RM1 in its local copy.
  b) If the resource R status is busy then the Activity A1 will block and wait for the next time tick to check and request.
  c) If the resource is available or becomes available, then F1 will send a request notification to occupy the resource R to F2 at time T2.
  d) If Federate F1 does not receive any resource request from Federate F2 at T2 it will update the status of RM1.
  e) Similarly, F2 will update the status of RM2 as busy/unavailable in its local copy of the resource and both the federates progress to next time.
  f) Once the activity in F1 has completed its operation and will not require the resource at time T3, it will send a Zero "0" in its request to indicate to F2 that F1 no longer requires the resource and F2 can mark the resource as now available.
  g) Both F1 and F2 will update the status of their local copies of resource at time T3.

h) Similarly, F2 can notify F1 to use the resource by following the above steps.

- **Simulation runtime Case B:**
  a) If the activities A1 and A2 in both models F1 and F2 wish to use resource R at the same time, then both federates will send request notifications to each other at time T1.
  b) Both models will evaluate the priority by looking at their priority table and update their local copy of the resource accordingly. In this case, a higher priority is assigned to Federate F1; therefore, Activity A1 in Federate F1 will occupy the resource R while M2 will wait for the resource to be available and block its activity in A2.
  c) After the resource is allocated the simulation will continue further: from Case A step f onwards, as given above.
  d) Once the resource becomes available, F2 must request for the resource again, which could lead to either Case A or Case B, until the simulation ends.

5.2.5.2 Next Request Protocol

The second possible approach to address IRM Type B.1 General Shared Resource is by using a Next request approach, i.e., if the resource is already busy then the resource will be booked for next use by any model. The relationship between the Models, CSP and RTI is shown in Figure 5.20 above. The concept of sharing resource R is the same as described in the last section, i.e., a shared resource is not owned by any federate or model; therefore, a copy of the same resource is kept at each model RMx. In this protocol explanation, two federates share the same resource.

This protocol may be applicable to a scenario similar to one explained in the previous section where two units in a factory may need attention either at fixed or variable time intervals and require a machine engineer/ operator to acknowledge them. Similarly, a resource could be represented as an object with multiple attributes, which could define more than one behaviour of the resource. But for simplicity, this research assumes only one behaviour of the resource as is in the case of the previous protocol. This protocol is further enhanced by having two parameters. Therefore, the resource within this protocol is represented as a single interaction class with one boolean parameter “RequestFx” for each federate (“x” represents the federate number), “DurationFx”.

The state of the request parameter (RequestFx) could be either zero “0” indicating no resource request or one “1” for resource requests. Duration parameter is used to indicate the duration in time when the resource will be occupied by the federate, i.e., it defines the time when the next resource request can be raised. This will allow the other federate to schedule its next event or wait without verifying at every time tick. Finally, the Model ID parameter can be used if there are more than two models in a given scenario, it will ease the identification of the source model. Unlike the simple protocol, each federate will maintain a boolean status variable in their local copies of resource requests (i.e., RMx), and a duration variable (D) to hold the time for which the resource will be occupied by another federate (i.e. Dx). The RMx status variable will be used to record the status of the Resource at any given time, which will be consistent across the federation. This variable will be null if there is no simultaneous resource request. Following FOM is used for this protocol.

```
<interactions>
    <interactionClass>
        <name>HLAinteractionRoot</name>
        <sharing>PublishSubscribe</sharing>
        <dimensions>NA</dimensions>
        <transportation>HLAreliable</transportation>
        <order>Receive</order>
        <interactionClass>
            <name> SharedResource </name>
            <sharing>PublishSubscribe</sharing>
            <transportation>HLAreliable</transportation>
            <order>TimeStamp</order>
            <parameter>
                <name> requestF1 </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name> requestF2 </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name> DurationF1  </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
             <parameter>
                <name> DurationF2  </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
        </interactionClass>
    </interactionClass>
</interactions>
```

In this protocol, Federate F1 will publish the "requestF1" and "DurationF1" parameters and subscribe to "requestF2" and "DurationF2" parameters of the interaction class, while Federate F2 will publish the "requestF2" and "DurationF2" parameters and subscribe to "requestF1" and "DurationF1" parameters of the interaction class. Each federate will use a fixed / static priority table in case both models request the same resource at the same time T. Again, to maintain consistency, Federate F1 will have a higher priority for using the resource over Federate F2.

In this protocol, each simulation model uses normal distribution and all the models are based on a stochastic approach. Both simulation models, i.e., Model M1 and

Model M2 use the same normal distribution for generating new entities (i.e., Arrivals) where the mean = 4.0 and Standard deviation SD = 1.5 (i.e. μ=4, σ = 1.5). The service time for Activity A1 and Activity A2 are also defined by Normal Distribution of mean = 4.0 and Standard deviation SD = 1.5 (i.e., μ=4, σ = 1.5). However, during simulation runs different values of mean and SD will be used. Figure 5.23 illustrates the sequence diagram of the proposed approach based on the protocol explained above. Figure 5.24 illustrate the flow of the protocol algorithm.

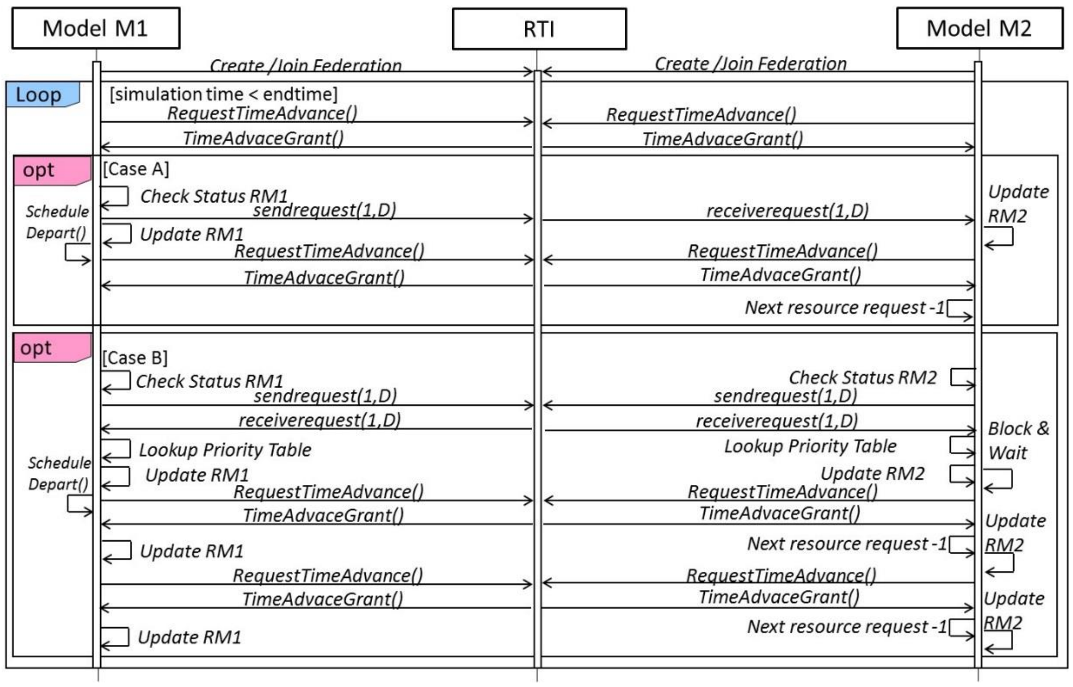


Figure 5.23: SD of General Shared Resource using Next Request

Sequence of Events

- **Initialisation stage:** Both Federates will either create or join federation.
- **Initialisation stage:** Federate F1 will indicate that it is capable of sending resource requests by publishing to "requestF1" and "DurationF1" and can receive by subscribing to "requestF2" and "DurationF2" interaction class parameters.

- **Initialisation stage:** Federate F2 will indicate that it is capable of sending resource requests by publishing to “requestF2” and “DurationF2” and can receive by subscribing to “requestF1” and “DurationF1” interaction class parameters.
- **Simulation Runtime:** Both Federates F1 and F2 will progress time “t” using the time management service of RTI.
- **Simulation runtime Case A:**

a) If Activity A1 in Federate F1 requires a resource at time T1 it will first check the status of the resource RM1 in its local copy.

b) If the resource R status is busy then the Activity A1 will block and schedule the next activity processing when the resource will be available next using time identified in duration variable D1.

c) If the resource is available or becomes available, then F1 will send a request notification to F2 to occupy the resource R at time T1, with the duration D1 required to process the entity.

d) If the Federate F1 does not receive any resource request from federate F2 at T1 it will update the status of RM1 (as available) and keep the federate name as null.

e) Similarly, F2 will update the status of RM2 as busy/unavailable in its local copy of the resource and update next available time for the resource. Both the federates progress to next time.

f) Once the activity in F1 has completed its operation and no longer requires the resource for the next activity, it will then send a Zero “0” in its request to indicate to F2 that F1 no longer requires the resource and F2 can mark the resource as now available.

g) Both F1 and F2 will update the status of their local copies of the resource at Time T2.

h) Similarly, F2 can notify to use Resource to F1 by following the above steps.

i) While the resource was blocked by F1, if F2 needed to use the resource it would send the resource request with duration to F1. F1 would also update the next request and occupy the resource once it had completed its processing.

- **Simulation runtime Case B:**

a) If the activities A1 and A2 in both models F1 and F2 wish to use resource R at the same time, then both federates will send request notifications to each other at time T1.

b) Both models will evaluate the priority by looking at their priority table and updating their local copy of the resource accordingly. In this case, a higher priority is assigned to Federate F1; therefore, Activity A1 in Federate F1 will occupy the resource R while M2 will wait for the resource to be available and block Activity in A2.

c) Also, as Federate F2 requested the resource at the same time, there will be no re-election for the resource and both models will update their federate name variable with "F2" as next requesting federate.

d) Once the Activity A1 has completed its operation and wants to release the resource, it will send a message to F2 at T2 and mark the resource status as unavailable and empty the next federate name variable.

e) When F2 receives the resource release message from F1 it will also update the next federate name with a null and keep the resource busy for its own use at T2.

f) Once Federate F2 has released the resource at T4, the resource is again available for use for both federates. But if at T3 F1 had requested the resource again, both the models would update their federate name variable with the next requesting federate and continue from Case B Step d.

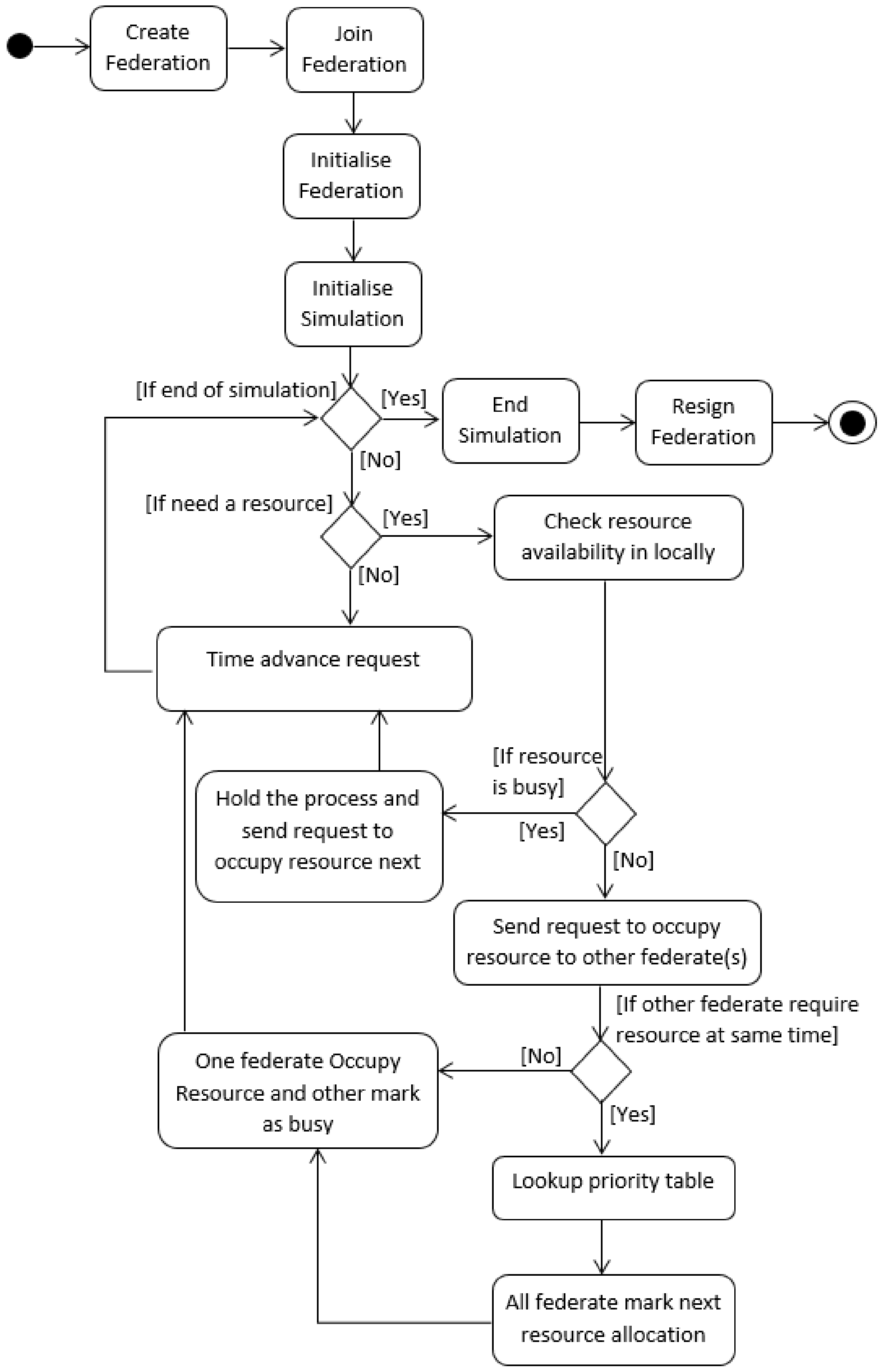

Create Federation
Join Federation
Initialise Federation
Initialise Simulation
[If end of simulation]
[Yes]
End Simulation
Resign Federation
[No]
[If need a resource]
[Yes]
Check resource availability in locally
[No]
Time advance request
[If resource is busy]
Hold the process and send request to occupy resource next
[Yes]
[No]
Send request to occupy resource to other federate(s)
[If other federate require resource at same time]
[No]
One federate Occupy Resource and other mark as busy
[Yes]
Lookup priority table
All federate mark next resource allocation

Figure 5.24 Activity Diagram for General Shared Rsrouces

5.2.5.3 Message Queue Protocol

The third possible approach to address IRM Type B.1 general shared resource is by using a message queue and a resource manager, i.e., each model will have a local resource manager (LRM) that will arrange the resource requirements of the models sharing resource. The relationship between the Models, CSP, and RTI is shown in Figure 5.25 below. Similarly, to the previous two approaches, a copy of the same resource is kept at each model RMx and the resource is shared between two models.

This approach for resource sharing between federates, where there is one resource shared by two or more federates without any predefined priority, can also be termed as a fairer distribution of resources. This could be based on FIFO (First In First Out), LIFO (Last in First Out) or any other policy. Any resource shared by two or more models is problematic for consistently maintaining the state of that resource in a distributed simulation. This model assumes that all the participating federates are publishing and subscribing.

In this approach, to maintain the consistent state of the resource, each federate is equipped with a LRM, which will control the resource allocation of its federate. When any activity A wants to use a resource R, it will send a request to its LRM to release the resource. The resource manager will check and release the resource to the activity A at appropriate time T. If the resource is not currently available and is being used by another federate then LRM will send a wait message to its activity until the resource is available to use. This protocol provides resource guarantee and avoids starvation.

This protocol could also be used to avoid a deadlock situation because of resource locking. This could be achieved by sending an extra dynamic priority variable in the communication, which will sort the resources according to the priority. Before allocating the priority, all conditions for deadlock need to be identified. Later, based on the deadlock algorithms, the resource dynamic priority could be assigned. Being dynamic gives the flexibility to reduce or increase the resource use by any federate. For example, if more than two federates are used (i.e. F1, F2 and F3) and some federates are dependent on other federates' output, i.e., if Activity A2 in Federate

F2 has requested the use of resource but does not have all its entities to process as they have to come from Activity A1 and A3 from Federates F1 and F3, which are also waiting for the resource to process. In such case, A1 in Federate F1 should have the highest priority of resource R1 followed by Federate F3 before allocating the resource to Federate F1. Similarly, during the same simulation run, if A1 has already produced an entity and already requested the resource to process the next entity (this can happen if A1 processing time is shorter than A2), while A2 is still waiting to process its previous entity because of resource availability, then A2 might have the highest priority.

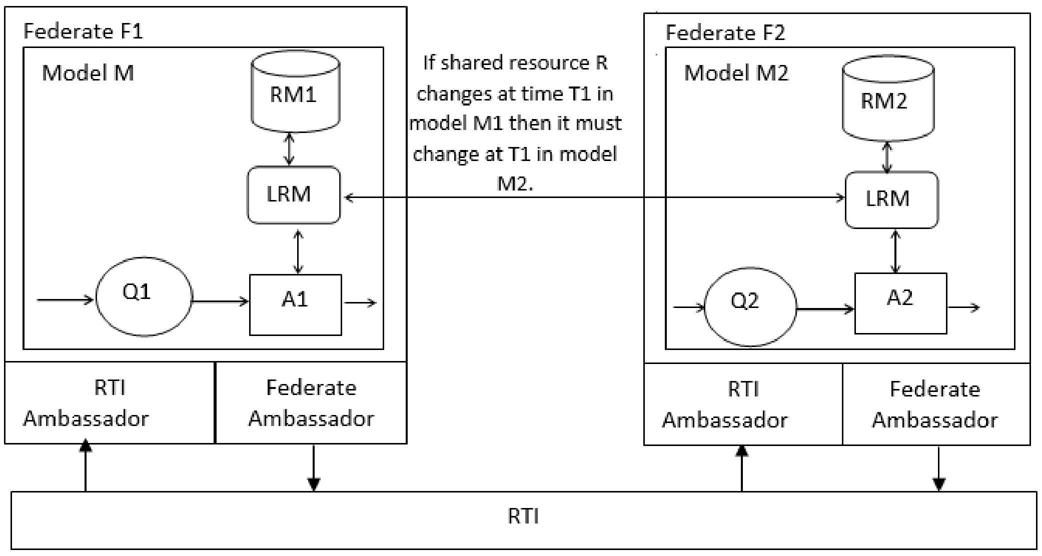


Figure 5.25: General Shared Resource with LRM using HLA

Message Queue

A message queue used in this proposed approach is a linked list where each node will be composed of the following as described in Figure 5.26

1. Federate name (FN)
2. Next release time (NRT)

Next release time will be calculated based on the following two cases:

Case A: If the message queue is empty

Next release time = Current simulation time T+ time required by activity before it could release the resource (AT).

Case B: If the message queue already has some previous notifications

The ***next release time*** = the last node of the message queue NRT + time required by activity before it could release the resource (AT).

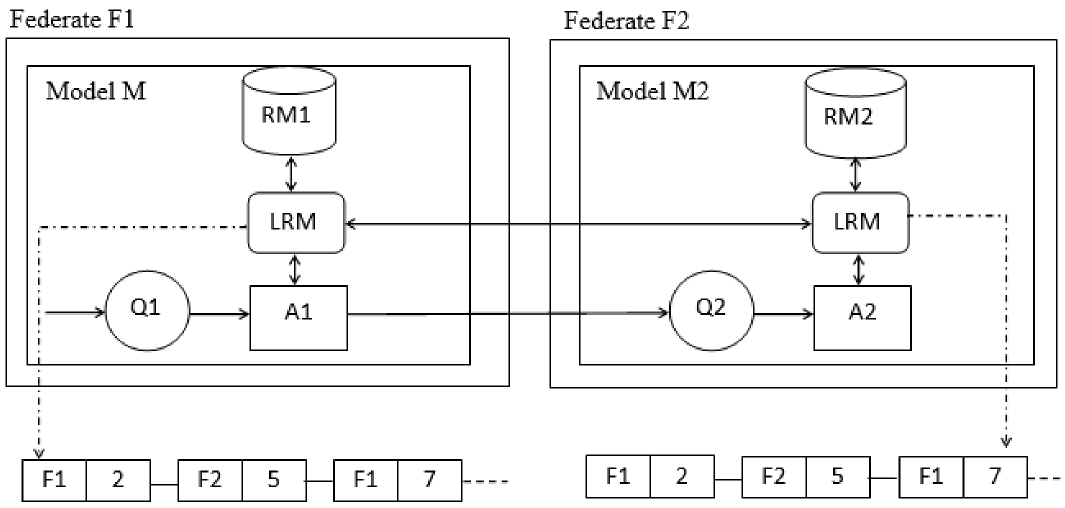


Figure 5.26: General Shared Resource with LRM using Message Queue

To share the state of the resource among all copies of federates a resource could be represented as an object and perform more than one role. Therefore, in this example, resource is represented as a single interaction class with one parameter "NRTFx" for each federate and "ModelIDFx". The NRT parameter will carry the time Activity will require to block the resource while ModelID will contain the federate name for requesting the resource. ModelID parameter is used because there are more than one models in a given scenario; therefore, it will ease the identification of the source model and will help in calculating the next resource available. Each federate will maintain a LRM (defined below) to store the status of the shared resource. The FOM used for this protocol is defined as:

```
1.  <interactions>
2.      <interactionClass>
3.          <name>HLAinteractionRoot</name>
4.          <sharing>PublishSubscribe</sharing>
5.          <dimensions>NA</dimensions>
6.          <transportation>HLAreliable</transportation>
7.          <order>Receive</order>
8.          <interactionClass>
9.              <name> SharedResource </name>
```

```
            <sharing>PublishSubscribe</sharing>
            <transportation>HLAreliable</transportation>
            <order>TimeStamp</order>
            <parameter>
                <name> NRTF1 </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name> NRTF2 </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name> ModelIDF1  </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
             <parameter>
                <name> ModelIDF2  </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
        </interactionClass>
    </interactionClass>
</interactions>
```

Using the above FOM, Federate F1 will publish the "NRTF1" and "ModelIDF1" parameter and subscribe to "NRTF2" & "ModelIDF2" parameter of the interaction class, while the SOM for Federate F2 will publish the "NRTF2" and "ModelID2" parameter and subscribe to "NRTF1" & "ModelID1" parameter of the interaction class. Each federate will also use a fixed / static priority table in cases when both models request the same resource at the same time T. Federate F1 will have higher priority for using the resource over Federate F2. Again, this priority table could also be dynamic or with a different strategy based on the model scenario and its requirements.

Local Resource Manager (LRM)

Each LRM will replicate the resource utilisation information to all other federates to maintain consistency. LRM is implemented by a message queue for each resource. This message queue is based on FIFO policy and it will store the federate name (FN) that is using or waiting to use the resource and the time taken by the activity to release the resource "NRT". If Activity wants to use a resource within Federate, it will send a resource release request to its LRM. LRM then will publish the Activity resource request (with request timestamp in microseconds, federate name, and the time required by Activity before releasing the resource) and wait for notification from other federate(s) before fulfilling the time advance request and before releasing the resource. If the resource is available, then the LRM will release the resource to Activity based on the following cases, otherwise it will ask Activity to wait until the resource is available.

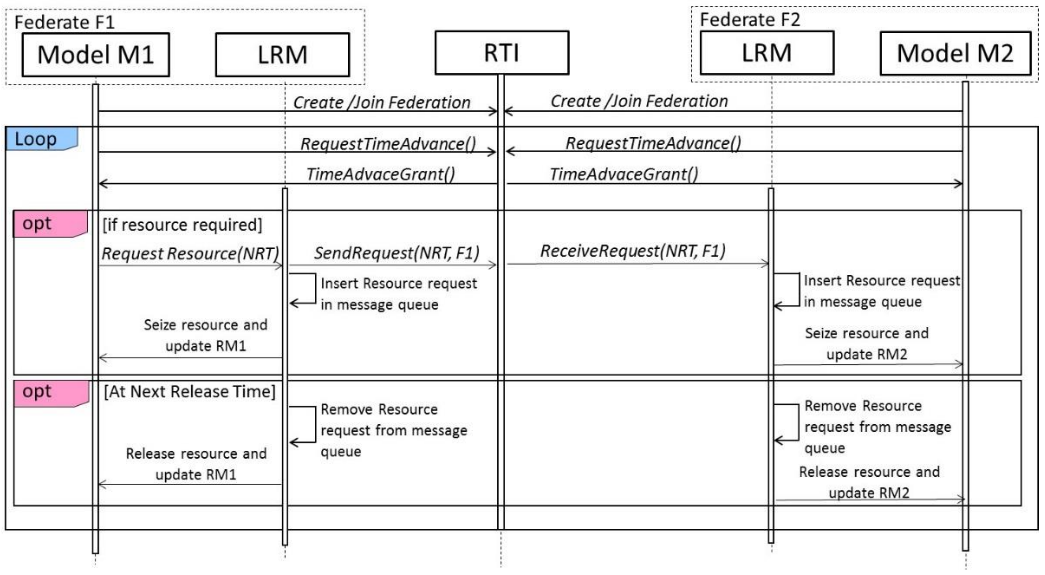


Figure 5.27: SD of General Shared Resource for Next Request

In this protocol each simulation model uses normal distribution and all the models are based on a stochastic approach. Both simulation models, i.e., Model M1 and Model M2 use the same normal distribution for generating new entities (i.e., Arrivals) where the mean = 4.0 and Standard deviation SD = 1.5 (i.e., μ=4, σ = 1.5). The service time for Activity A1 and Activity A2 are also defined by Normal Distribution of mean = 4.0 and Standard deviation SD = 1.5 (i.e. μ=4, σ = 1.5). However, during simulation runs different values of mean and SD will be used. Figure 5.27 illustrates the sequence diagram of the proposed approach based on the protocol explained above. This diagram illustrates an example of how messages are communicated between LRM and the model. Figure 5.24 illustrate the flow of the protocol algorithm.

Sequence of Events

- **Initialisation stage:** Both Federates will either create or join federation**.**
- **Initialisation stage:** Federate F1 will indicate that it is capable of sending a resource request by publishing to "NRTF1" and "ModelIDF1" and can receive by subscription to "NRTF2" and "ModelIDF2" interaction class parameter.
- **Initialisation stage:** Federate F2 will indicate that it is capable of sending a resource request by publishing to "NRTF2" and "ModelIDF2" and can receive by subscription to "NRTF1" and "ModelIDF1" interaction class parameter.
- **Simulation Runtime:** Both Federate F1 and F2 will progress time "t" using the time management service of RTI.
- **Simulation runtime Case A**
  a) If Activity A1 in Federate F1 requires a resource, then Federate F1 will publish the notification at a specific time T1 and send T2 (NRT) to Federate F2.
  b) All other federate LRMs will update their message queue with the published notification.
  c) The LRM in F1 will check against its local message queue if there is no previous notification, then the LRM will release the resource to A1 and request time advance.
  d) If there is a previous notification in the queue (i.e., the resource is busy and is not available) then it will ask Activity A1 to wait until time T2 and store the notification in the message queue.
  e) At time T2 the LRM will remove a notification from the message queue if NRT is equal to the current simulation time. The resource will be then released once all the previous notifications are timed out.

- **Simulation runtime Case B**

  a) If both federates have published the notification at the same simulation time, then the LRM of each federate will prioritise the notification from all federates, based on the local requested timestamp and it will insert the new information in their local message queue. Each LRM will check against the first request in their local message queue and next time release. Further execution will continue as discussed in case A.

  b) Note: The priority decision is not made on the order of messages received by the federate, instead it is based on the ascending TSO to accommodate both TSO and received order (RO) federate message configuration.

- **Simulation runtime Case C**

  a) If other federates have also published the notification at the same simulation time with the same timestamp this will mean messages having the same timestamp will be delivered in an arbitrary order (i.e., no tie-breaking mechanism is provided by an RTI) (IEEE 1516.2-2010).

  b) In such a case, when two or more notifications have the same timestamp, then the messages could be prioritised. Similar to the previous approaches, the resource will be prioritised by ascending order of the federate name to make sure all federates have a consistent message queue, i.e., in this case, it will be Federate F1 with higher priority. Further execution will continue as discussed in case A.

Case C uses the hybrid Specialised and dynamic prioritisation technique, which is much more flexible and scalable. Having priority set using federate name will determine the priority order at run time, while the strategy is defined specifically to all models.

5.2.5.4 Next Resource Event Protocol

The fourth possible approach to address IRM Type B.1 general shared resource is similar to the one discussed in the last section, but instead of using a message queue all the requests for resources are recorded in an event list. This event list already exists in the discrete event simulation approach; therefore, list management functions are not necessarily required to be separately implemented. The relationship between the Models, CSP and RTI is shown in Figure 5.20. Similar to the last three approaches, a copy of the same resource is kept at each model RMx and the resource is shared between two models.

To share the state of the resource in this protocol, a single interaction class with one parameter "NRT" is defined. The NRT parameter will carry the time required by the activity to use a resource and also this will be used to calculate when to schedule the next event in the event list. Both the Models will maintain a local variable Last next request time "NRT". This variable will be updated on each resource request locally. This variable helps to determine the next resource allocation and release time. In this approach, when Activity A1 from model M1 requires a resource, both model M1 and Model M2 will block the resource and at the same time T they will schedule an event to release the resource at the time defined by NRT. Note that in the discrete event simulation implementation the event list implements time based strategy for its event. But our sequential scheduling of the event will make sure that the resource request is allocated using FIFO strategy and Activity A1 will use the resource and Activity A2 will wait until NRT. The FOM used for this implementation is as under.

```
<interactions>
    <interactionClass>
        <name>HLAinteractionRoot</name>
        <sharing>PublishSubscribe</sharing>
        <dimensions>NA</dimensions>
        <transportation>HLAreliable</transportation>
        <order>Receive</order>
        <interactionClass>
            <name>SharedResource</name>
            <sharing>PublishSubscribe</sharing>
            <transportation>HLAreliable</transportation>
            <order>TimeStamp</order>
            <parameter>
                <name> NRTF1 </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name> NRTF2 </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
        </interactionClass>
    </interactionClass>
</interactions>
```

Using the above FOM, Federate F1 will publish the “NRTF1” parameter and subscribe to “NRTF2” parameter of the interaction class, while federate F2 will publish the “NRTF2” parameter and subscribe to “NRTF1” parameter of the interaction class. Each federate will also use a fixed / static priority table in cases when both models request the same resource at the same time T and Federate F1 will have higher priority for using the resource over Federate F2.

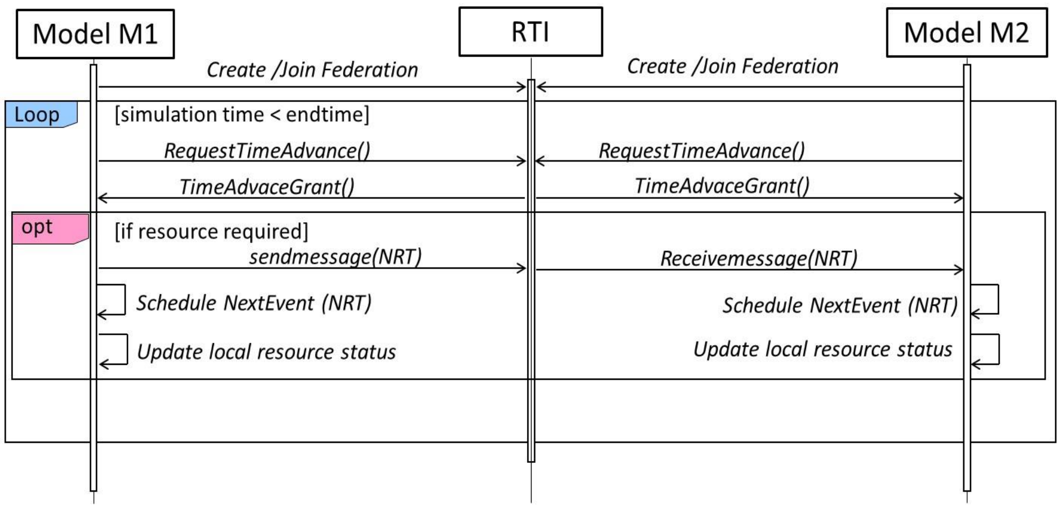


Figure 5.28: SD of General Shared Resource using Next Resource Event

Each simulation model uses normal distribution and both simulation models, i.e., Model M1 and Model M2 use the same distribution for generating new entities (i.e., Arrivals) where the mean = 4.0 and Standard deviation SD = 1.5 (i.e. μ=4, σ = 1.5). The service time for Activity A1 and Activity A2 are also defined by Normal Distribution of mean = 4.0 and Standard deviation SD = 1.5 (i.e. μ=4, σ = 1.5). However, during simulation runs different values of mean and SD will be used. Figure 5.28 illustrates the sequence diagram of the proposed approach based on the protocol explained above. This diagram illustrates an example of how messages are communicated between LRM and the model. Figure 5.24 illustrate the flow of the protocol algorithm.

Sequence of Events

- **Initialisation stage:** Both Federates will either create or join federation**.**
- **Initialisation stage:** Federate F1 will indicate that it is capable of sending resource requests by publishing to “NRTF1” and can receive by subscribing to “NRTF2” interaction class parameter.
- **Initialisation stage:** Federate F2 will indicate that it is capable of sending resource requests by publishing to “NRTF2” and can receive by subscribing to “NRTF1” interaction class parameter.
- **Simulation Runtime:** Both Federate F1 and F2 will progress time “t” using the time management service of RTI.
- **Simulation runtime Case A**
  a) If Activity A1 in Federate F1 requires use of a resource, then Federate F1 will publish the notification at a specific time T1 and send T2 as NRT to Federate F2.
  b) Both Model M1 and Model M2 will mark the resource as busy in their local copy of RMx and schedule an event to release the resource using the NRT.
  c) Since the resource is blocked by Model M1, if Activity A2 in Model M2 wishes to use the resource, then it has to repeat the request procedure from Step a.
  d) The resource will be booked using the last NRT time plus the time required for activity A2 to schedule the activity A2 and to release the resource.
- **Simulation runtime Case B**
  a) If both federates have published the resource request at the same time, then the predefined priority strategy must be applied. In this case Federate F1 has higher priority, therefore two events will be scheduled for using the resource, but the first event will be for Model M1 and later for Model M2.

b) Both Models will mark the resource as busy in their local copy.

5.2.6 General Shared Event

IRM Type C.1: General Shared Event represents an interoperability problem that occurs when two or more models in the distributed simulation are sharing the same events at the same time. It has been established in Chapter 4 that each model will have its local copy of the shared event, i.e., the process that will execute once the shared event is triggered. Therefore, a shared event message (a trigger) will be sent to all participating models, in which this event will not join any queue but will immediately trigger the shared event. It is also possible that these events are not same for all models, but they all have to start at the same time. For example, evacuating the building (when a fire alarm is triggered) might be quicker for people living on the ground floor, while people living on the second floor have to perform additional tasks and a process of climbing down the stairs.

Figure 5.29 illustrates the relationship between the Model, CSP and RTI. For the purpose of understanding the general shared event concept, it is necessary to identify the participating models, i.e., all models that will send and receive the shared event message, and are required to process the shared event at time T. Therefore, it will be assumed that two models, M1 and M2 to participate in the shared event example and Time t will be defined as the time when the event is triggered and the same time as the shared event starts processing at both models. For the purpose of simplicity M1 will trigger the shared event at time T1 and both models will run the shared event EM1 and EM2 at time T2 (where T1=T2=t).

The event trigger to execute the shared event, in both Model M1 from Federate F1 and M2 from Federate F2, could be represented as a single variable message or an object with attributes. There is also expected to be multiple types of triggers and shared event processes in real practice. For example, an industrial assembly line can have more than one production unit feeding to one packaging unit. If one production unit breaks down or slows down when others are still working then an event message can be sent to the packaging unit to reduce the packaging speed by some factor and the information about which unit is broken down, etc. Therefore, this information could be sent in the event trigger object as attribute or interaction values. The actual representation of the event trigger message depends on the model implementation in a CSP and how the modeller will use the message.

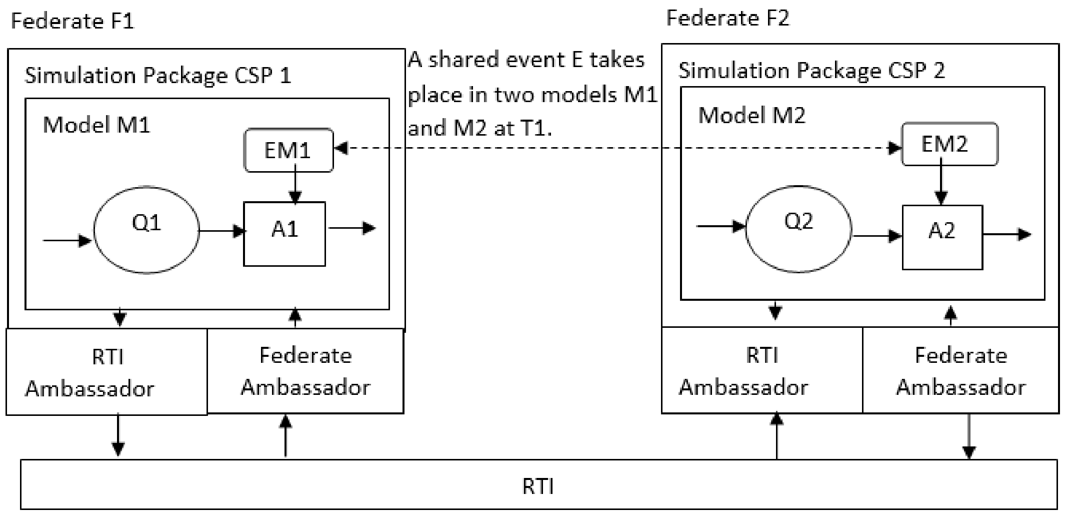


Figure 5.29: General Shared Event HLA

For the purpose of understanding the event trigger message used, it is represented as a single interaction class parameter "Event" in both models, M1 and M2. As there are only two models in a selected scenario, both models will be aware of the message source. However, defining the source of the event is important in case of more than two models used and when there are different types of event triggers and/or different shared event procedures. This "Event" attribute of the interaction class will be Published and Subscribed by both models by using RTI features. Therefore, both models will use the following FOM.

```
1.  <interactions>
2.      <interactionClass>
3.          <name>HLAinteractionRoot</name>
4.          <sharing>PublishSubscribe</sharing>
5.          <dimensions>NA</dimensions>
6.          <transportation>HLAreliable</transportation>
```

```
        <order>Receive</order>
        <interactionClass>
            <name>SharedEvent</name>
            <sharing>PublishSubscribe</sharing>
            <transportation>HLAreliable</transportation>
            <order>TimeStamp</order>
            <parameter>
                <name> event </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
        </interactionClass>
    </interactionClass>
</interactions>
```

Using the above FOM, Federate F1 will publish the “event” parameter, while Federate F2 will subscribe to “event” parameter of the interaction class. Each federate will have its local copy of the shared event and each simulation model uses normal distribution. Both the models, i.e., Model M1 and Model M2 use the same distribution for generating new entities (i.e., Arrivals) where the mean = 4.0 and Standard deviation SD = 1.5 (i.e. μ=4, $\sigma = 1.5$). The service time for Activity A1 and Activity A2 are also defined by Normal Distribution of mean = 4.0 and Standard deviation SD = 1.5 (i.e. μ=4, $\sigma = 1.5$). However, during simulation runs an event is generated with a normal distribution where the mean = 100 and Standard deviation SD = 15 (i.e. μ=100, $\sigma = 15$). Figure 5.30 illustrates the sequence diagram of the proposed approach based on the protocol explained above. This diagram illustrates an example of how messages are communicated between RTI and the model and Figure 5.31 illustrate the flow of the protocol algorithm.

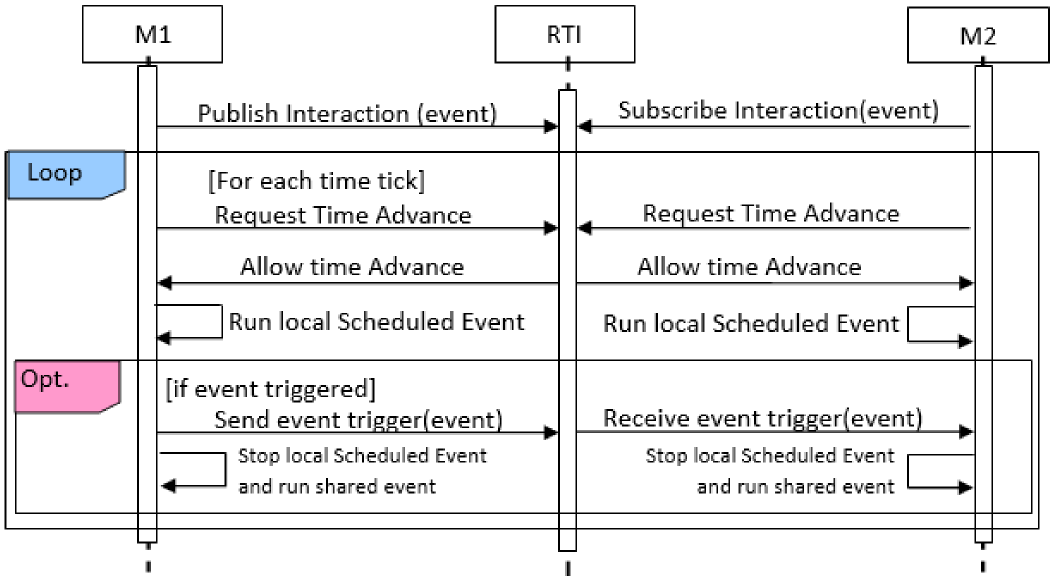

M1
RTI
M2
Publish Interaction (event)
Subscribe Interaction(event)
Loop
[For each time tick]
Request Time Advance
Request Time Advance
Allow time Advance
Allow time Advance
Run local Scheduled Event
Run local Scheduled Event
Opt.
[if event triggered]
Send event trigger(event)
Receive event trigger(event)
Stop local Scheduled Event and run shared event
Stop local Scheduled Event and run shared event

Figure 5.30: SD for a General Shared Event using RTI

Sequence of Events

- **Initialisation stage:** Both Federates will either create or join federation**.**
- **Initialisation stage:** Federate F1 will indicate that it is capable of sending an event message by publishing "event" interaction class parameter.
- **Initialisation stage:** Federate F2 will indicate that it is capable of receiving an event message by subscribing to "event" interaction class parameter.
- **Simulation Runtime:** Both Federates F1 and F2 will progress time "t" using the time management service of RTI and schedule their local events.
- **Simulation runtime Case A**

  a) If Model M1 in Federate F1 requires triggering a shared event at time T1, then Federate F1 will broadcast the notification at T1 to all available federates.

  b) Both Model M1 and Model M2 will stop their current activities and schedule a local shared event process.

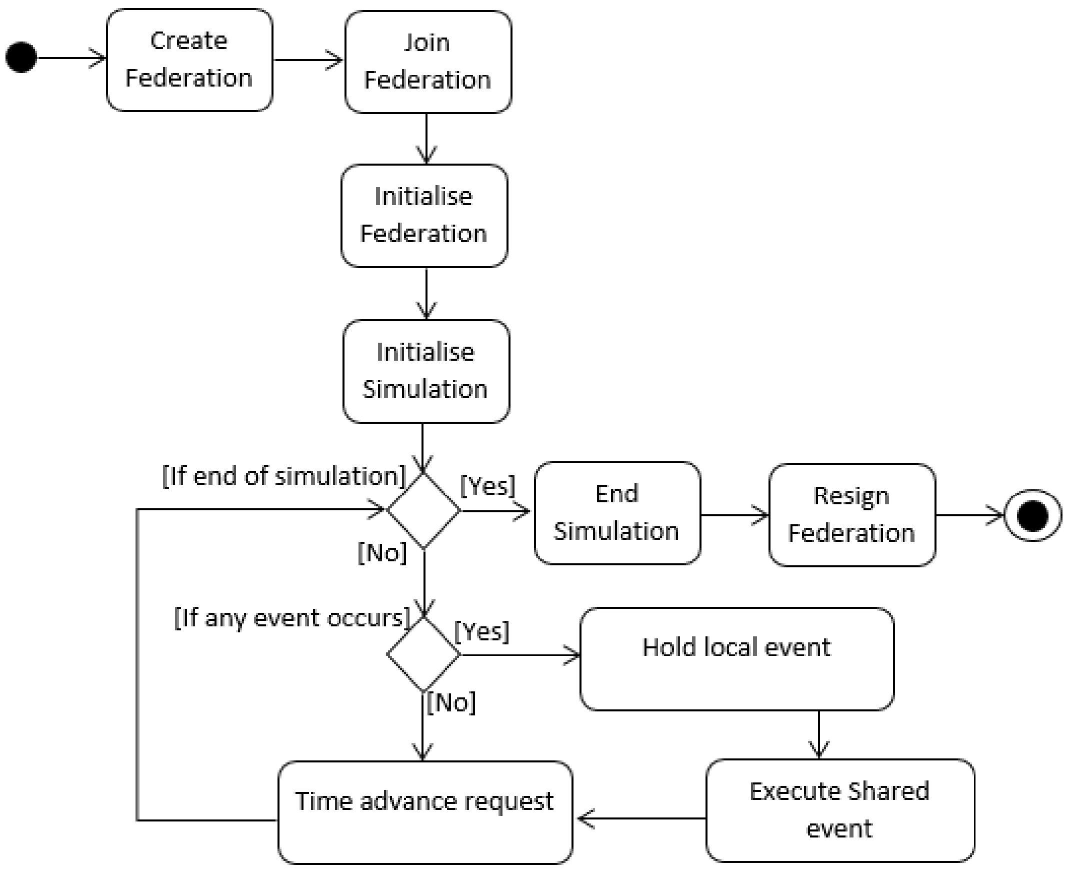


Figure 5.31: Activity Diagram for Shared Event

5.2.7 General Shared Data Structure

IRM Type D: Shared Data Structure represents an interoperability problem for sharing data between two or more models in a distributed simulation environment as described in Chapter 4. Although a shared data structure is not the same as sharing resources, there are some similarities; therefore, it must not be confused with shared resources. Shared resources face challenges like time guarantee, avoiding deadlocks, and resource starvation, while a shared data structure comes with challenges like consistency, deadlock, concurrent access, and update anomaly. This problem is already defined in Chapter 4 in more detail, and therefore this section will focus on the conceptual modelling of each protocol used to address this IRM.

Distributed simulation consists of different models, each having some event. An event is an action occurring at a certain time t. An event could be related to access a shared data, i.e., a request to read or update some data value. Any read request is to get the latest available value of the data structure. An update is a process where a data variable is first read, then a certain operation or event is applied to generate a new value which is used to replace the old value of that same variable. These event sequences in a distributed simulation are performed according to the TSO associated with them. All requests having the same timestamp are executed either in a sequential order or by event priority according to the program code of the event. The sequential execution of events might lead to a different problem. For example, if a model M1 requested a read operation at same time when another model M2 requested an update operation of shared data structure D, then by following sequence inconsistent data would be returned to model M1, when the actual sequence of events was to change and update the shared data before the read of the shared data structure D.

Therefore, one approach to implement a shared data in a distributed simulation environment is to use a DSM model on top of the distributed simulation application. This DSM acts as a virtual physical memory for all of the models. In simplicity, all models will have one shared memory externally. However, DSM does not fully address the issues in distributed simulation applications and creates further issues related to casualties (Mehl and Hammes, 1995). Therefore, this approach is modified below to best suit the requirements for distributed applications. Based on this DSM model, the following three approaches could be considered.

1) Central control

In this approach a shared variable or data structure is kept on a separate node (federate). This node will be the owner of the variable and will retain the most updated copy of the variable. This node will receive requests for both read and update and will make decisions on request before sending the information back. Additional processes need to be implemented to avoid bottlenecks, casualties, and to maintain consistency while dealing with multiple shared variables. Alternatively, each shared variable could be hosted by different nodes.

2) Central Update

In this approach a copy of shared variable or data structure is kept at each model (requesting shared data). The value of these copies will be kept consistent at all times, allowing immediate concurrent access to read the value; updates will be controlled by the central server.

3) Full replication

In this approach, a copy of shared variable or data structure is kept at each model (requesting shared data). The values of these copies will be kept consistent at all times allowing immediate access to read the values, but the updates will also be controlled by these models. This means no central server will be implemented to control read and update as all relevant processes will be implemented within the models. This approach is more flexible and scalable, but with costs to complexity and efficiency.

5.2.7.1 Central Control Protocol

The first possible approach to address IRM Type D.1 General Shared Data Structure is by using the central control approach as explained in previous section. The relationship between the models, CSP and RTI is illustrated in Figure 5.32. For the purpose of understanding the Shared Data Structure with the central control approach, it is necessary to identify responsibilities and how the models are sharing the data structure. In this approach, a shared data structure is not owned by the requesting federate, instead a separate federate, termed as controller federate, will maintain the concurrent state of the data structure and the requesting federate will request a copy of the data structure at every read or write using the Data Manager (DM) in controller federate.

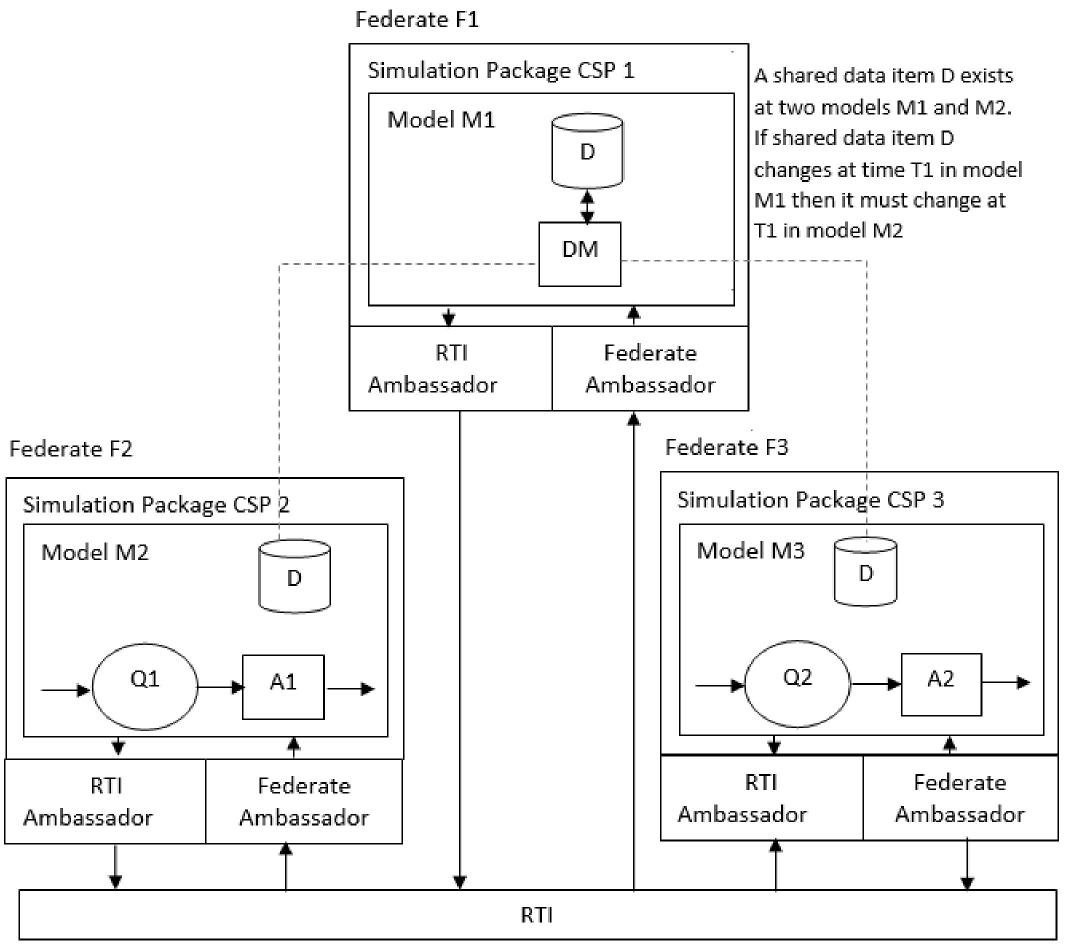


Figure 5.32: Shared Data Structure using Central Control & Update approach using HLA

To maintain consistency among all federates for shared data structure at any given time t the controller federate will use an exclusive locking mechanism whereby access to the data structure could be restricted for other federate(s) while an update is in process.

A controller federate can implement more than one shared data structure or more than one controller federate can be created for each shared data structure but the concept of access will remain the same. In this protocol there is one controller federate F1 with one shared data structure between two other requesting federates, F2 and F3.

To share the state of the data structure among all the federates a shared data structure could be represented as an object with a number of attributes as required by the model. Shared data structure in this protocol is represented as a single interaction class with three parameters: a string parameter for the federate name or ID “ModelID” for each federate identification; a string parameter “opcode” to define the operation, i.e., Read or Write (R or W); and the “value” to send the new value. The value parameter will be null in case of the read operation. The following FOM will be used by all three federates.

```
<interactions>
    <interactionClass>
        <name>HLAinteractionRoot</name>
        <sharing>PublishSubscribe</sharing>
        <dimensions>NA</dimensions>
        <transportation>HLAreliable</transportation>
        <order>Receive</order>
        <interactionClass>
            <name>SharedData</name>
            <sharing>PublishSubscribe</sharing>
            <transportation>HLAreliable</transportation>
            <order>TimeStamp</order>
            <parameter>
                <name> valueF1 </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name> ModelIDF2</name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name> opcodeF2</name>
                <dataType>HLAASCIIchar </dataType>
            </parameter>
            <parameter>
                <name> valueF2 </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name> ModelIDF3</name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name> opcodeF3 </name>
                <dataType>HLAASCIIchar </dataType>
            </parameter>
            <parameter>
                <name> valueF3 </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
        </interactionClass>
    </interactionClass>
</interactions>
```

In the above FOM, Federate F2 will publish the “ModelIDF2”, “opcodeF2” and “valueF2” parameter and subscribe to “valueF1” parameter of the interaction class from Federate F1. Similarly, Federate F3 will publish the “ModelIDF3”, opcodeF3” and “valueF3” parameter and subscribe to “valueF1” parameter of the interaction class from Federate F1. While Federate F1 will publish the “valueF1” parameter and subscribe to “ModelIDF2”, “opcodeF2”, “valueF2”, “ModelIDF3”, “opcodeF3” and “valueF3” of the interaction class.

Each federate will use a fixed / static priority table in cases when two or more models initiate an update request for the shared data structure at the same time t. In this case, Federate F2 has higher priority for updating the shared data structure over Federate F3. Again, this priority table could also be dynamic or with a different strategy based on the model scenario and requirements.

Read Request

If a read request is initiated on shared data structures by the requesting federates at time T1, this request (including ModelID, opcode, value) is sent to Federate F1, while the requesting federates suspend operations until the controller returns the value. The controller will hold these read requests until receipt guarantee assures it will not receive any further update request. When the controller is satisfied of this, it extracts the latest value of the data structure at time T1 and returns to the requesting federates. In read only operation the value parameter of the request will be null.

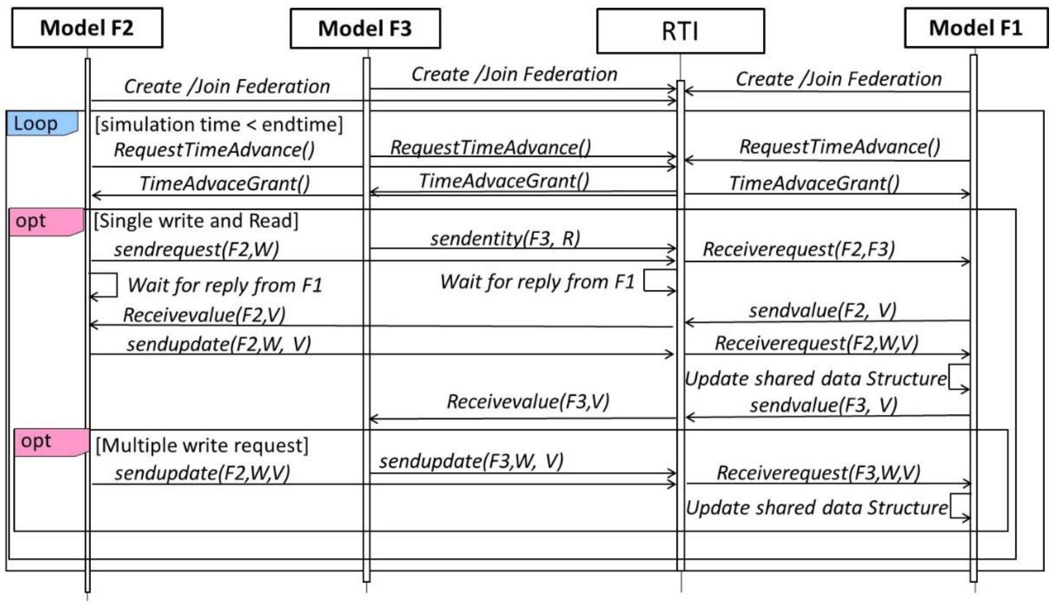


Figure 5.33: SD for Shared Data Structure using Central Control

Update Request

If an update request is initiated on shared data structures by the requesting federates at time T1, this request is sent to the controller Federate F1. While the requesting Federates F2 and F3 will suspend their operations until the controller returns the value. The controller will hold all the requests until receipt guarantee ensures it will not receive any further update request. If the controller receives a single update request, then it will send the current shared data structure value to the federate requesting an update and hold the reply to other federates (if requesting to read). The federate requesting an update will send an updated value to the controller and the controller will then send this new value to the other federates (if requested). But if in case two or more federates have requested an update, then this could be prioritised either by using a timestamp approach or defined priority approach as discussed in other IRM solutions explained above. Note, the update operation means data has to be read first before an operation is done to update the value. The sequence diagram in Figure 5.33 above illustrates the sequence of events and Figure 5.34 illustrate the flow of the protocol algorithm.

Sequence of Events

- **Initialisation stage:** All Federateswill either create or join federation**.**
- **Initialisation stage:** Federate F2 and Federate F3 will indicate that they are capable of sending requests by publishing “ModelIDF2”, “opcodeF2”, “valueF2”, “ModelIDF3”, “opcodeF3” and “valueF3” interaction class parameters respectively. While Federate F1 will publish “valueF1” interaction class parameter.
- **Initialisation stage:** Federate F2 and Federate F3 will indicate that they are capable of receiving requests by subscribing to “valueF1” interaction class parameter while Federate F1 will indicate that it is capable of receiving requests by subscribing to “ModelIDF2”, “opcodeF2”, “valueF2”, “ModelIDF3”, “opcodeF3”, and “valueF3”.
- **Simulation Runtime:** All federates F1, F2 and F3 will progress time “t” using the time management service of RTI.
- **Simulation runtime Case A (Read only):**
  a) If Activity A1 in Federate F2 requires reading the shared data at time T1 it will send a read request (ModelID F1, opcode R and value V) to F1 at time T1 and will wait for a reply.
  b) Once Federate F1 has received all the requests from other federates and there is no update request in the list, then Federate F1 will reply to all read requests with the latest copy of the shared data structure D using the value (V) to Federate F2 and Federate F3 (if requested).
  c) All federates will progress to next time.

- **Simulation runtime Case B (single update):**
  a) If Activity A1 in Federate F2 requires an update to the shared data at Time T1 it will send an update request to F1 at time T1 and will wait for other federate(s) to send their request.
  b) Once Federate F1 has received all the requests from other federates and there is no other update request in the list, then Federate F1 will send the current value to Federate F2 at time T1 and will hold the reply to another read request (if any).
  c) Federate F2 will update the variable and send the new value to Federate F1. Federate F1 will replace the shared data structure value with the new value before acknowledging the read requests from other federates (if any).
  d) All federates will progress to next time.

- **Simulation runtime Case C (Multiple updates):**
  a) If Activity A1 and Activity A2 in Federate F2 and Federate F3 require an update to the shared data D at the same time T1, they will send an update read request to F1 at time T1 and will wait for other federate(s) to send their request.
  b) Once Federate F1 has received all the requests, then based on priority (as discussed above) F2 will receive the current value of the shared data structure to update while Federate F3 will hold its operation and will wait until the value is not being updated.
  c) When Federate F1 receives the new value (V) from Federate F2, it will send it to Federate F3 for its update. Federate F3 will process the value and generate a new value (V) which will be posted to Federate F1 at same time T1. Federate F1 will update the shared data structure D.
  d) All federates will time advance.

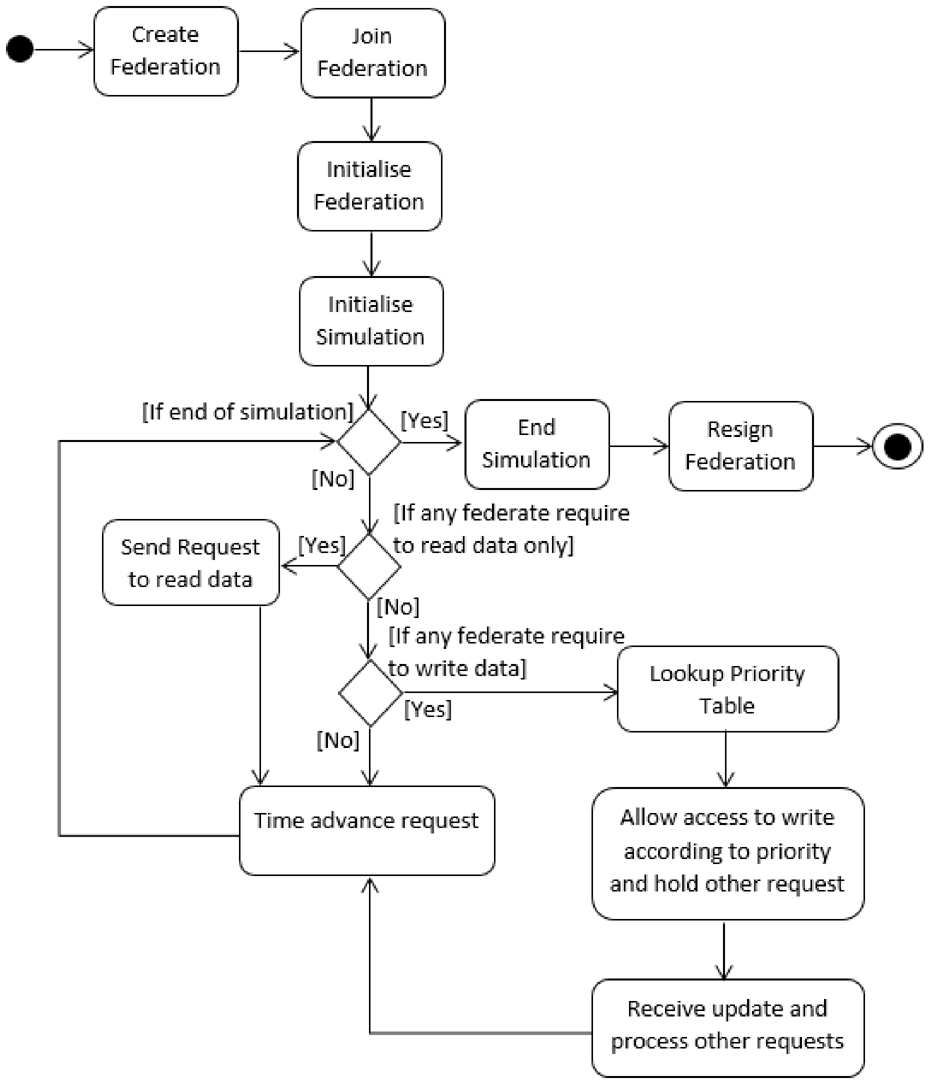

Create Federation
Join Federation
Initialise Federation
Initialise Simulation
[If end of simulation]
[Yes]
End Simulation
Resign Federation
[No]
[If any federate require to read data only]
Send Request to read data
[Yes]
[No]
[If any federate require to write data]
Lookup Priority Table
[Yes]
[No]
Time advance request
Allow access to write according to priority and hold other request
Receive update and process other requests

Figure 5.34: Activity Diagram for Central Control Protocol

5.2.7.2 Central Update Protocol

The second possible approach to address IRM Type D.1 General Shared Data Structure is by using the central update approach as defined above. The relationship between the models, CSP and RTI is illustrated in Figure 5.32. In this approach, as discussed above, a copy of the data is kept at the concerned federates to have concurrent access for any read operation while the updates are managed by a central server. Unfortunately, this approach cannot directly be used in every scenario in the conservative simulation approach because rollback is not possible. For example, suppose M2 requests a read to its local data at time t. Further, M2 proceeds with execution until at any virtual time tv it requests to read data again. Because its read-only copy of data is still not updated, M2 would continue to read and use the data from its local copy, whereas in real time the model might be messaged about a new value update by another federate M3, which would invalidate the read copy of M2 at virtual time tt, where $t < tv = tt$. Therefore, the read request at tv by M2 can use an invalid copy. Being a conservative simulation, the algorithm does not offer rollback or undo wrong event executions, so this approach can compromise consistency. However, this approach could be used within simulation where the lookahead is greater than zero, the simulation exercises a rollback facility, or the frequency of use of shared data is minimized.

Some arrangements must therefore be made whereby a local copy remains valid until the next time advance when the federate is notified about the change in state of the data, while the read operation continues without any further communication. Additional strategies could be defined based on model requirements.

This approach could also represent a situation where the update operations are not the responsibility of the requesting federate and the update is done by some other federate. For example, in a military war game, firing a missile from an aircraft or firing a shell from a tank could represent two different federates while the effect of these two different arsenals hitting targets could only be calculated by a third federate responsible for terrain impact. Both other federates could send parameters to define the power of their arsenals, but the effects have to be calculated and updated to both federates by the terrain federate. Both the requesting federates can then hold the updated terrain condition as a local copy for their use.

In this approach, a shared data structure is partially owned by the requesting federate, along with a third federate which will be responsible for updating and maintaining the concurrent state of the data structure. The requesting federate will request an update using the DM in the third federate. To maintain consistency of shared data structure among all federates at any given time t, the third federate will use an exclusive locking mechanism to update the data structure whereby access to the data structure could be restricted for other federate(s) while an update is in process. In this protocol, unlike the last central control approach, DM will take the values as parameters from the other federates and will process further to generate a new value before it sends out to the requesting federates. There is one federate F1 responsible for updates, with one shared data structure between two other requesting federates, F2 and F3.

To share the state of the data structure among all federates, a shared data structure could be represented as an object with a number of attributes as required by the model, e.g., a shared data could be an object with listed attributes. Shared data structure in this approach is represented by a single interaction class with two parameters: a string parameter for the federate name “Model ID” for each federate identification; and an integer parameter “value” to provide the value for the update. The FOM used by the federates is defined below.

```
<interactions>
    <interactionClass>
        <name>HLAinteractionRoot</name>
        <sharing>PublishSubscribe</sharing>
        <dimensions>NA</dimensions>
        <transportation>HLAreliable</transportation>
        <order>Receive</order>
        <interactionClass>
            <name>SharedData</name>
            <sharing>PublishSubscribe</sharing>
            <transportation>HLAreliable</transportation>
            <order>TimeStamp</order>
            <parameter>
                <name> valueF1 </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name> ModelIDF2</name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name> valueF2 </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
             <parameter>
                <name> ModelIDF3</name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
            <parameter>
                <name> valueF3 </name>
                <dataType>HLAinteger32BE</dataType>
            </parameter>
        </interactionClass>
    </interactionClass>
</interactions>
```

Using the above FOM, Federate F2 will publish the “ModelIDF2” and “valueF2” parameter and subscribe to “valueF1” parameter of the interaction class from Federate F1. Similarly, Federate F3 will publish the “ModelIDF3” and “valueF3” parameter and subscribe to “valueF1” parameter of the interaction class from Federate F1. Federate F1 will publish the “valueF1” parameter and subscribe to “ModelIDF2”, “valueF2”, “ModelID3” and “valueF3” of the interaction class.

Federate F1 will use a fixed / static priority table in cases when two or more models initiate an update request for the shared data structure at the same time t. In this example, Federate F2 has higher priority for updating the shared data structure over Federate F3. Again, this priority table could also be dynamic or with a different strategy based on model scenario and requirements.

Read Request

If a read request is initiated on shared data structure by requesting federates at time t, this request need not to be sent to Federate F1, as the requesting federate will use their local copy of the shared data structure for this purpose.

Update Request

If an update request is initiated on shared data structure by requesting federates at time T1, this request is sent to Federate F1 and the requesting federate will use the read operation from its local copy of the shared data structure. Federate F1 will hold all requests until by receipt guarantee it knows it will not receive any further update request. If Federate F1 receives a single update request, then it will update the shared data structure only, but if two or more federates have requested an update, then this could be prioritised either by using a timestamp approach or defined priority approach, as discussed in other IRM solutions explained above. Once the data is updated, a new value will be sent to all requesting federates at the same time T1 before time advance. The sequence of events is illustrated by the sequence diagram below in Figure 5.35.

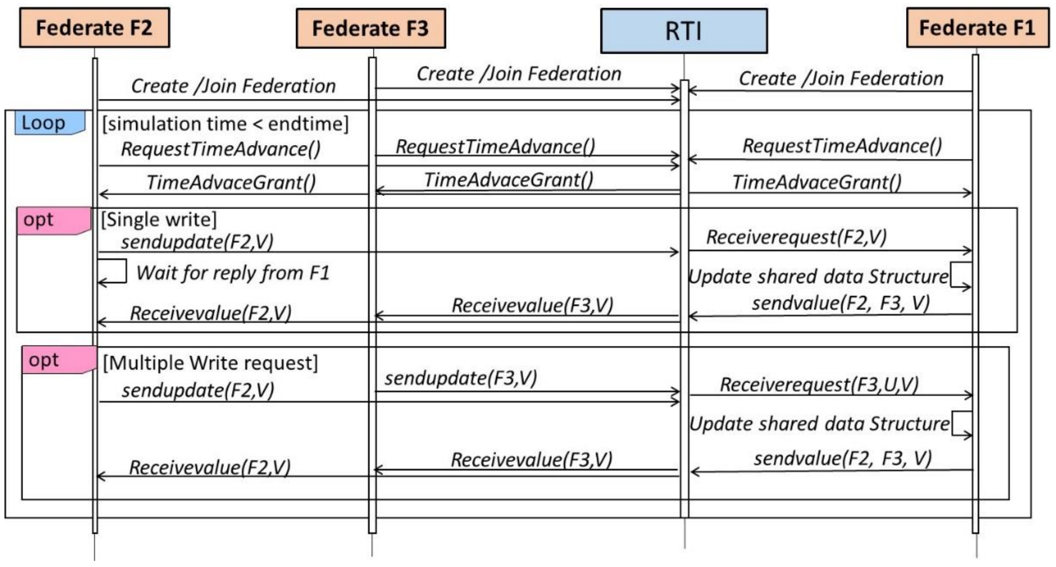


Figure 5.35: Sequence Diagram of Central Update Shared Data Structure

Sequence of Events

- **Initialisation stage:** All Federates will either create or join federation.
- **Initialisation stage:** Federate F2 and Federate F3 will indicate that they are capable of sending an update request by publishing "ModelIDF2", "valueF2", "ModelIDF3" and "valueF3" interaction class parameter respectively to Federate F1. While Federate F1 will publish "valueF1" interaction class parameter.
- **Initialisation stage:** Federate F2 and Federate F3 will indicate that they are capable of receiving requests by subscribing to "valueF1" interaction class

parameter while Federate F1 will indicate that it is capable of receiving requests by subscribing to “ModelIDF2”, “valueF2”, “ModelIDF3” and “valueF3”.

- **Simulation Runtime:** All Federates F1, F2 and F3 will progress time “t” using the time management service of RTI.
- **Simulation runtime Case A (Read only):**

  a) If Activity A1 in Federate F2 or Activity A2 in Federate F3 require to read the shared data at time T1 they will not send any message to Federate F1 and will use their local copy of shared data structure D.

  b) All federates will progress to next time.
- **Simulation runtime Case B (single update):**

  a) If Activity A1 in Federate F2 requires an update to the shared data at time T1, then it will send an update request (federate ID F2, and value V) to F1 at time T1 and will wait for other federate(s) to send their request.

  b) Once Federate F1 has received all requests from other federates and there is no other update request in the list, then Federate F1 will use the parameter value (V) to update the current value of the shared data structure and send the current updated value (value V) to Federate F2 and Federate F3 at time T1.

  c) Federate F2 and Federate F3 will update their local copy of shared data structure D at time T1.

  d) All federates will progress to next time.
- **Simulation runtime Case C (Multiple updates):**

  a) If Activity A1 in Federate F2 and Activity A2 in Federate F3 require an update to the shared data at time T1, then they will send an update request to F1 at time T1.

  b) Once Federate F1 has received all requests from federates and there is no other update request in the list, Federate F1 will use the parameter value to update the current value of the shared data structure based on the priority set (in this case, F2 parameter will be used first to update the value).

c) Once both updates are applied by Federate F1, Federate F1 will send the current value to Federate F2 and Federate F3 at time T1.

d) Federate F2 and Federate F3 will update their local copy of shared data structure at time T1.

e) All federates will progress to next time.

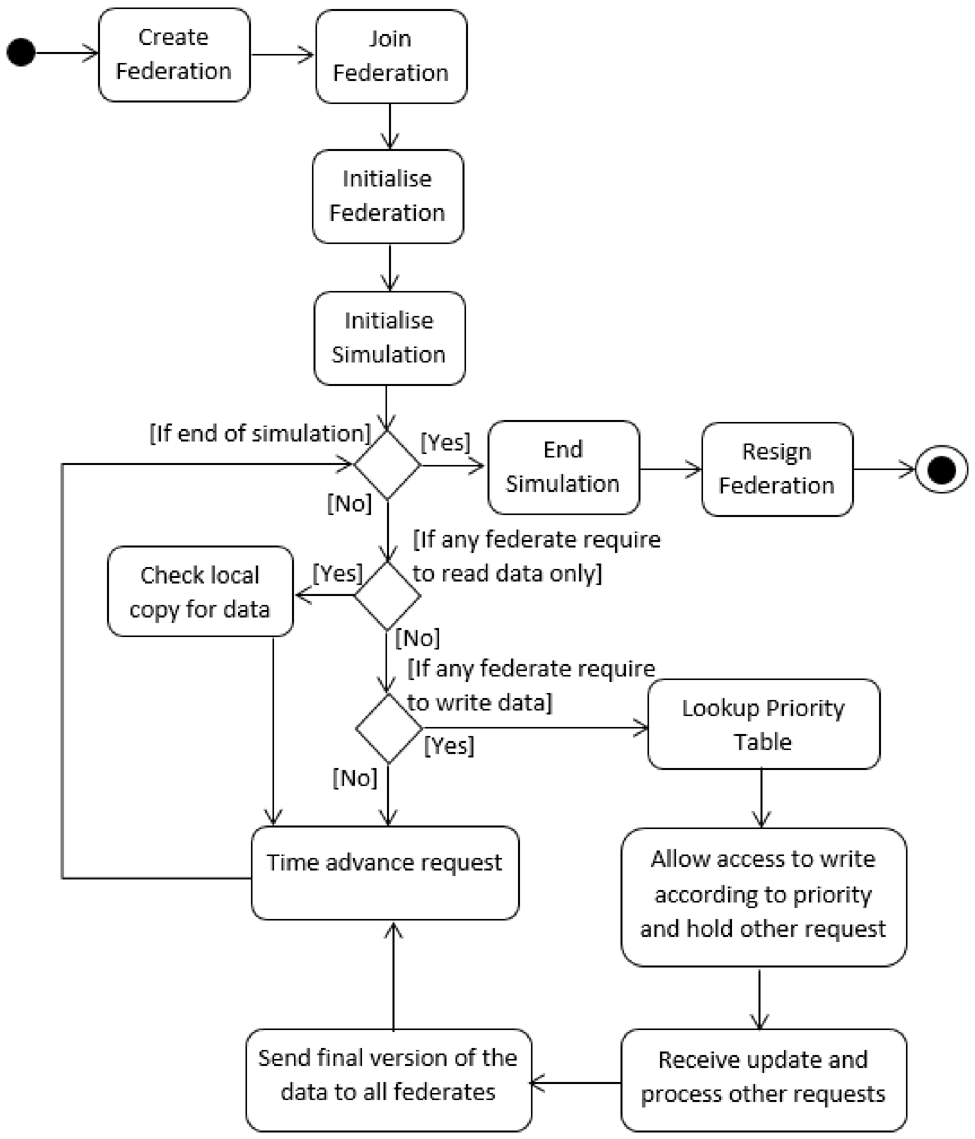

Create Federation
Join Federation
Initialise Federation
Initialise Simulation
[If end of simulation]
[Yes]
End Simulation
Resign Federation
[No]
[If any federate require to read data only]
[Yes]
Check local copy for data
[No]
[If any federate require to write data]
[Yes]
Lookup Priority Table
[No]
Time advance request
Allow access to write according to priority and hold other request
Send final version of the data to all federates
Receive update and process other requests

Figure 5.36: Activity Diagram for Central Update Protocol

5.2.7.3 Full Replication Protocol

The third possible approach to address IRM Type D.1: General Shared Data Structure is by using a full replication approach, as defined above. There is no need for a third federate to manage the shared data as the shared data copies are kept with each federate; therefore, a minimum of two federates are required to demonstrate the full replication behaviour. The relationship between the models, CSP, and RTI is illustrated in Figure 5.37. In conservative simulation, full replication can be implemented more effectively with non-zero lookahead. To block the read request before the correct value is updated is a challenge in full replication and can only be possible if the update request is known before a read operation. In a zero lookahead scenario and in the case of multiple updates with no blocking to read operations, increased overheads for the update could result. It could mean running update operations more than once, which will increase cost of time and complexity, but only for the concurrent update. No communication is required for read operations. This protocol illustrates how full replication can be achieved using zero lookahead with multiple updates.

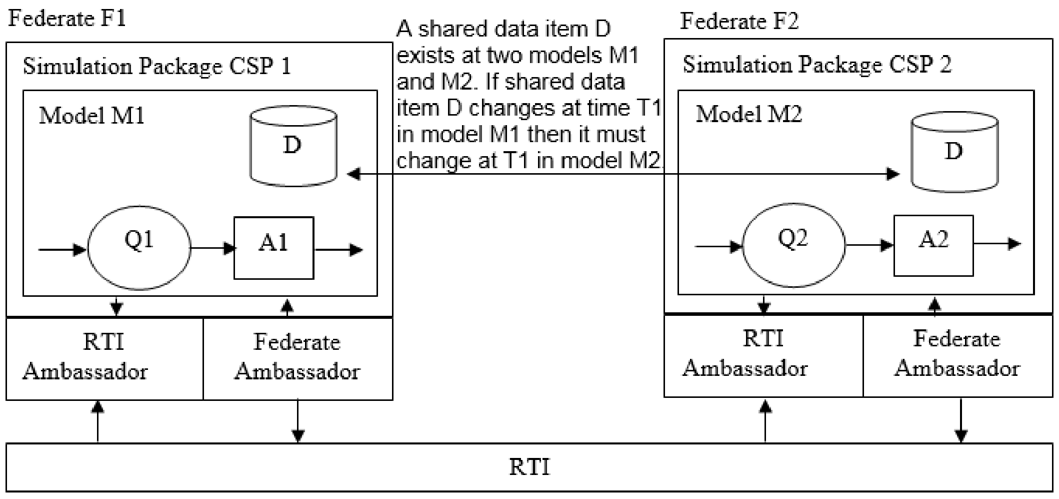


Figure 5.37: Shared Data Structure using Full Replication approach

In this approach, a shared data structure is completely owned by the requesting federate, and no third federate is involved in the process for updating and maintaining the concurrent state of the shared data structure D. To maintain consistency among all federates for shared data structure at any given time t, both federates will use message passing to update the state of their local copies of the shared data structure. Shared data structure for this protocol is represented as a single interaction class with one as an integer value “value” to provide the new value for the update. The FOM used for this protocol is as under.

```
<interactions>
  <interactionClass>
    <name>HLAinteractionRoot</name>
    <sharing>PublishSubscribe</sharing>
    <dimensions>NA</dimensions>
    <transportation>HLAreliable</transportation>
    <order>Receive</order>
    <interactionClass>
      <name>SharedData</name>
      <sharing>PublishSubscribe</sharing>
      <transportation>HLAreliable</transportation>
      <order>TimeStamp</order>
      <parameter>
        <name> valueF1 </name>
        <dataType>HLAinteger32BE</dataType>
      </parameter>
      <parameter>
        <name> valueF2 </name>
        <dataType>HLAinteger32BE</dataType>
      </parameter>
    </interactionClass>
  </interactionClass>
```

23. </interactions>

Using the above FOM, Federate F1 will publish "valueF1" parameter and subscribe to "valueF2" parameter of the interaction class from Federate F2. Similarly, Federate F2 will publish "valueF2" parameter and subscribe to "valueF1" parameter of the interaction class. Each federate will use a fixed / static priority table in cases when two or more models initiate an update request for the shared data structure at the same time t. A higher priority for Federate F1 is defined for updating the shared data structure.

Read Request

If a read request is initiated on shared data structure by requesting federates at time t, the requesting federate will use their local copy of the shared data structure for this purpose.

Update Request

If an update request is initiated on shared data structure by requesting federates at time T1, this request is sent to the other federate(s) sharing the data structure. The requesting federate will hold its operation until it receives a null value or no further update requests from other federates at time T1. If the first Federate has not received any other update request, then it will update the local copy of the shared data structure. The other federates will update their local copy of the shared data structure with the message received. But if two or more federates have requested an update at the same time T1, then this could be prioritised either by using a timestamp approach or defined priority approach, as discussed in other IRM solutions explained above. In a concurrent update request, the first selected federate will update its local copy of the shared data structure and ignore the value from the other federate. While the other federate will use the updated value of the first selected federate and re-run the update process to generate a new value. Alternatively, the event related to update can be scheduled after receiving the value from other federates. This new value will be sent to the first selected federate which will get updated in its local copy of the shared data structure while the first federate waits before both can advance time. The sequence of events is illustrated by the sequence diagram below in Figure 5.38. Figure 5.39 illustrate the flow of the protocol algorithm.

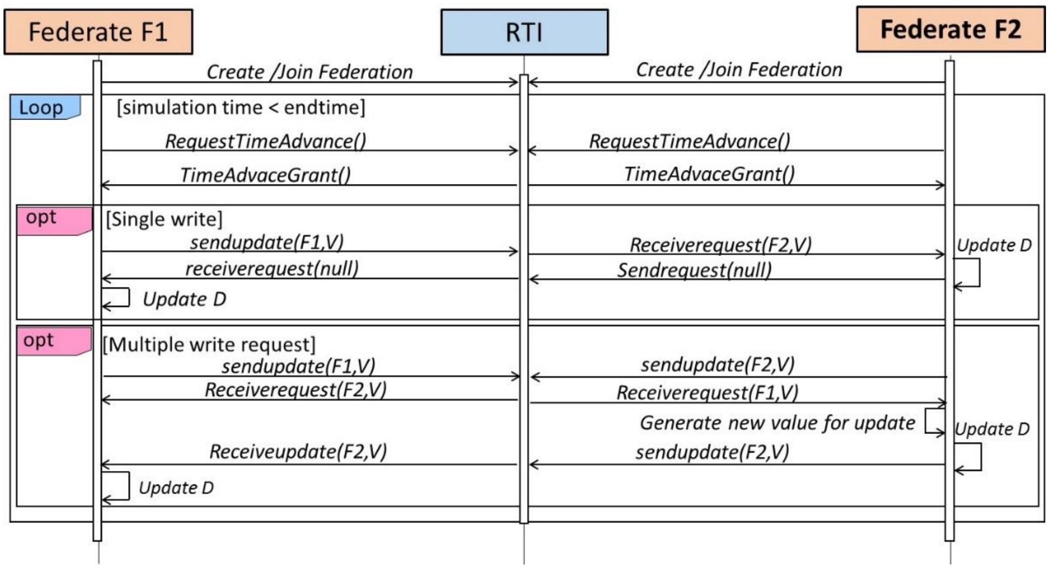


Figure 5.38: SD of Full Replication Shared Data Structure

Sequence of Events

- **Initialisation stage:** All Federates will either create or join federation.
- **Initialisation stage:** Federate F1 and Federate F2 will indicate that they are capable of sending an update request by publishing “FnameF1”, “ValueF1”, “FnameF2” and “valueF2” interaction class parameter respectively.
- **Initialisation stage:** Federate F1 and Federate F2 will indicate that they are capable of receiving requests by subscribing to “FnameF2”, “ValueF2”, “FnameF1” and “valueF1” interaction class parameter respectively.
- **Simulation Runtime:** Both Federates F1 and F2 will progress time “t” using the time management service of RTI.
- **Simulation runtime Case A (Read only):**

  a) If Activity A1 in Federate F1 or Activity A2 in Federate F2 require to read the shared data at time T1, they will not send any message to Federate F1 but will use their local copy of the shared data structure.

  b) All federates will progress to next time.

- **Simulation runtime Case B (single update):**

  a) If Activity A1 in Federate F1 requires an update to the shared data D at time T1, then it will send an update request (Federate name F1, and a new value V) to F2 at time T1 and will wait for other federate(s) to send their request.

  b) If Federate F1 receives no request from Federate F2 then Federate F1 will use the new value to update the current value of the shared data structure (D) and Federate F2 will update its local copy of the shared data structure D at time T1.

  c) All federates will progress to next time.

- **Simulation runtime Case C (Multiple updates):**

  a) If Activity A1 in Federate F1 and Activity A2 in Federate F2 require an update to the shared data at time T1, then they will send an update request to each other at time T1 with their new value (V).

  b) Federate F1 will use the new value to update its current value of the shared data structure D and wait for the reply from Federate F2 (based on the priority set, in case F1 value will be used first to update the value).

  c) Federate F2 will run its update operation again using the new value from Federate F1 and generate a new value.

  d) Federate F2 will then update its local copy of shared data structure (D) with this new value and also send this new value (V) to Federate F1 at time T1.

  e) Federate F1 will update its local copy of shared data structure using the new value at time T1.

  f) All federates will progress to next time.

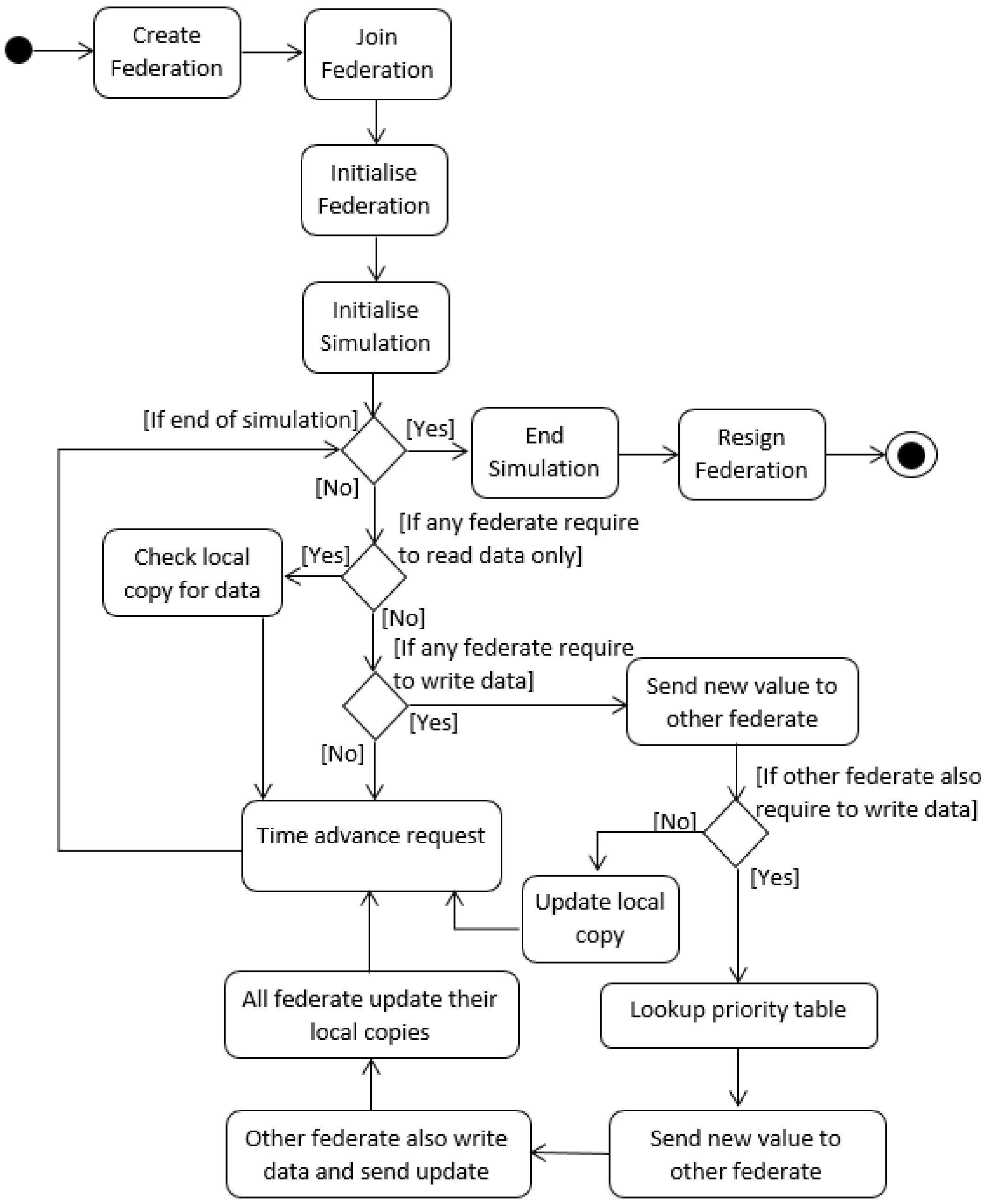

Create Federation
Join Federation
Initialise Federation
Initialise Simulation
[If end of simulation]
[Yes]
End Simulation
Resign Federation
[No]
[If any federate require to read data only]
[Yes]
Check local copy for data
[No]
[If any federate require to write data]
[Yes]
Send new value to other federate
[No]
Time advance request
[If other federate also require to write data]
[No]
[Yes]
Update local copy
All federate update their local copies
Lookup priority table
Other federate also write data and send update
Send new value to other federate

Figure 5.39: Activity Diagram Full Replication

5.3 Simulation Verification and Validation

To test the performance of the possible solutions to the interoperability issues identified in IRM, simple distributed simulation models were created with no model over heads but they all are time synchronised using RTI. All these models were created using HLA 1516e standard. These models will help benchmark the performance for each protocol defined above. The overall system V&V is achieved by addressing the following level of testing.

1) Member application testing: Each IRM defined in SISO-STD-006-2010 has some expectations and constraints; these are discussed in Chapter 4. During setting the experimental design, the requirements are reviewed for each IRM to make sure they comply with the conditions.

2) Integration & Interoperability testing: During this testing, data was collected at different points of simulation runs and verified against the required interoperability behaviour. Each protocol was initially executed with short runs. These runs were introduced with check points at which the data was debugged. This data was also scrutinised to verify the overall simulation.

5.3.1 Integration & Interoperability testing

The six IRM types implemented by a number of different protocols and each protocol consist at least a minimum of two federate. One of each federate act as a sender (i.e. The federate that generated its own entities and after processing send them to other federate), while the other federate act as a receiver (i.e. This federate do not generate its own entity, but depends on entity send by the sender federate). Both of these federate run in a single federation. Since these two federates are running on a separate machine to simulate a specific simulation scenario. There are issues related to Integration and Interoperability between them. To test this, there was a list of several check points introduced to verify each protocol, a sample is listed in Table 3. This table illustrates four check points for sender federate and three for receiver.

The example used in this table is for IRM type A.1: General Entity Transfer. The parameters include the time when the entity arrived (or generated in sender federate), followed by the time after the processing is completed on that entity and when the entity is ready to be send and finally, when the time the entity is sent. On the receiving side, similar parameters are monitored such as the time the entity received followed by the timely processing is completed. This table also monitors the queue size, because in some protocol's likes IRM A.2 Bounded buffer, it was important to note the behavior of receiving side queue.

Sender Federate				Receiver Federate		
Entity Arrival time	**Sender Queue size**	**Sender Process time**	**Entity Send time**	**Received Entities time**	**Receiver Queue Size**	**Receiver Process time**
0	0	1	1	1	0	2.5
2	0	3	3	3	0	4.5
7	0	8	8	8	0	9.5
11	0	12	12	12	0	13.5
11	1	13	13	13	1	15
15	0	16	16	16	1	16.5
16	0	17	17	17	2	18
16	1	18	18	18	2	19.5
19	0	20	20	20	1	21
20	0	21	21	21	2	22.5
22	0	23	23	23	2	24
25	0	26	26	26	1	25.5
. . . .	. . . .	. . . .	. . . .	. . . .	. . . .	. . . .
36900	**-**	**36600**	**36600**	**36600**	**-**	**36590**

Table 3: Integration & Interoperability testing

In case of IRM Type A.3: Multiple input prioritization and Type D General Shared resources, there was a minimum of three federate used, but with the same parameters. Since, the distribution is stochastic i.e. the entity generated by the sender have random probability distribution or pattern, therefore, it is important that the data must be captured from both federates in a single run.

The testing begins by running the model for a shorter simulation time, later the output from both federates were joined together and analysed for checking the behavior of the protocol. This was also verified and matched with each IRM requirements as defined above. The important part of the interoperability test is to check if the sending entities matches with the receiving entities. At times when the protocol did not had the desired output it was revised and a shorter simulation run was tested before running the federation for bigger simulation runs. However, the data compared identified that the models are behaving the same as they were if running in a non-distributed environment.

5.4 Case Study: London Emergency Medical Service (EMS)

Emergency Medical Services systems are heterogeneous, complex and multidimensional. All over the world they provide medical services to patients for minor or critical illnesses or injuries on site or during transport to the hospital. This research used a scaled down prototype model of London Emergency Medical Services (EMS) as an exemplar case study to measure the interoperability, scalability and the performance of some of the proposed protocols in a hybrid environment. The EMS system represents several Accident and Emergency (A&E) systems at various London hospitals. These A&E systems are grouped together with ambulance services, facilitating A&E departments.

The ambulance model consists of emergency call centre(s), the crew and vehicles. This is accessible to the public through an emergency telephone number. Before, dispatching the appropriate vehicle and crew, the call operator has to access the nature of the emergency over the phone. Later, the call operator has to find and dispatch the closest available vehicle and crew to the incident site. After administering first aid, the crew make a decision on whether the patient needs to be hospitalised or released after onsite treatment. If they make a decision to transfer the patient to a hospital, then the nearest available hospital is selected. After transferring the patient to the hospital, the ambulance and the crew will travel to its standby location and will be once again available for any future response.

The ambulance service performance is measured by three distinct periods in its service timeline as shown in Figure 5.40. The time an emergency call is made until the selection of the appropriate ambulance is measured by *waiting* time. Similarly, time of departure of the ambulance until the transfer of the patient to the hospital is the *service* time. The *response* time is measured from the time the call was made until the ambulance arrives at the scene. The boxes in the diagram indicate the events that will take place in the model.

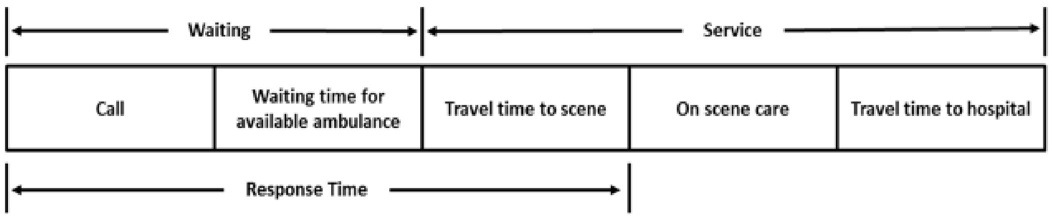


Figure 5.40: Ambulance Service timeline (Fitzsimmons, 1973)

Travel time for the ambulance is calculated assuming an average London ambulance speed and the Euclidean distance between the starting point and the ending point. According to Silva and Pinto (2010), "use of a corrective coefficient to realistically represent the relationship between the actual distance and the Euclidean distance of two points in an urban environment". Therefore, each location point has an X and a Y Cartesian coordinate, so the Euclidean distance |s| between two points is:

$$|s| = \sqrt{(X_A - X_B)^2 + (Y_A - Y_B)^2}$$

The travel time is: $t = c * s / v$, where t is the travel time, c is the corrective coefficient, s *is the distance and* v is the average speed. With the help of google maps a corrective coefficient was generated as $c = 1.32$. This was obtained from 40 samples, ranging from 0.5 to 20.5 miles across Greater London.

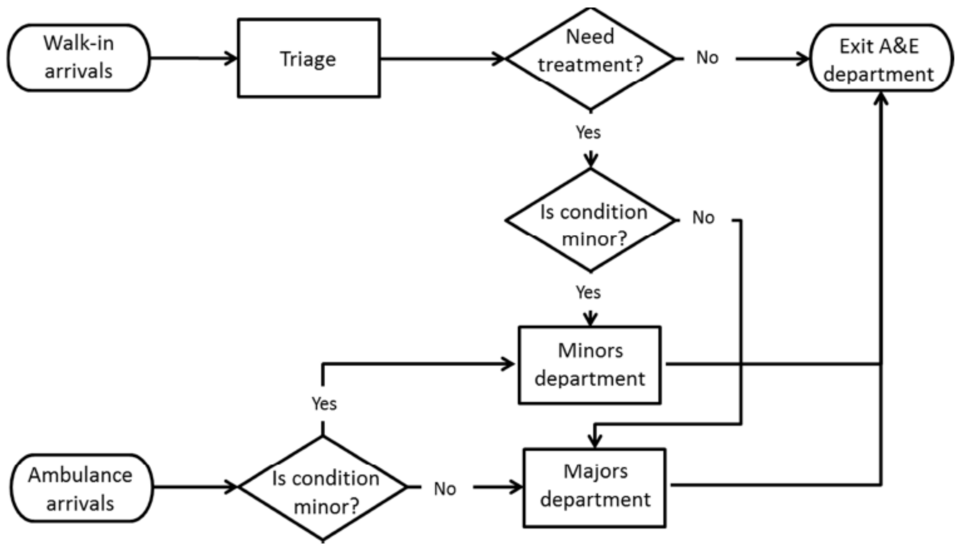


Figure 5.41: A&E department flowchart diagram (Anagnostou, 2014).

However, the A&E department in a hospital is in itself a complete and complex system. A&E department is not restricted to serve the patient brought in via ambulance services, but also for all other patients including walk-ins. This would mean that the resources such as doctors, cubicle or beds and equipment etc. varies. There are no number of fix resources allocated for any type of emergency. A decision is made based on the patient condition. If a patient arrives through ambulance service they are not treated as differently, if the required resources are not available then the patient had to wait. But in most of the cases the patient conditions and details are shared by the ambulance crew while they are selecting the hospital. This provides an advance notice to A&E department to prepare the resources based on the type of emergency. For the purpose of understanding, this case study has classified the emergency in two types, i.e. major or minor. Major mean it is a serious life-threatening situation while a patient with a minor condition could wait before they get any further treatment from A&E. Some patient might need further treatment, for which they can be either transferred to the hospital or send home with arrangements. In some cases, a patient could be referred to their GP for further treatment. The functioning of hospital and GP is not part of this case study, therefore they will not be referenced in further discussion. Further details of these models can be found in the full work of Anagnostou (2014).

Since A&E department is linked with other activates (such as specialist clinics, medical test facilities and hospital etc.), therefore the level of abstraction for this model were kept high. The main purpose was to demonstrate the interaction between Ambulance model and A&E and also how the proposed protocols could be used to provide interoperability solutions for such a problem. Therefore, all the models represent general functions of the A&E department without considering the resuscitation units in them. Figure 5.41 present a flowchart of two different scenarios where the patient arrives at the hospital through walk-in and through ambulance service. Walk-in patients had to go for a triage to identify the seriousness of the emergency, while in case of ambulance services, this is done by the ambulance crew while administering first aid. After this stage patient joins appropriate queue for treatment before exiting A&E department. While accessing, the availability of the hospital, clinical staff and beds/cubical are considered as follows:

Available clinical staff = staff capacity – occupied staff

And

$$\textit{Available beds}_{type} = \textit{capacity}_{type} - \textit{occupancy}_{type} \;\; \textit{:where type} = \{\textit{minor, major}\}$$

5.4.1 Data Collection

The The ambulance service prototype model in this case study is a 1:5 scaled-down London EMS system. Department of Health (DoH) is ultimate responsible for the London Ambulance service (LAS) through NHS trust. Therefore, the data used in this model were taken from the NHS England website during the year 2011 to 2012. According to the DoH, London ambulance service has a fleet of 998 vehicles, 375 of which are abundances covering 620mi^2. It was 70 ambulance station with 5 headquarters within the boundaries of the M25 (www.londonambulance.nhs.uk).

Similar to the ambulance service model, the data were also acquired from the same published sources for A&E department. Based on the data and identified process above, following list of events is generated in Table 4.

Ambulance Model	**A&E Model**
Emergency call	Patient arrival
Ambulance found	Triage Queue
Arrival at incident scene	Triage Service
Hospital found	Minors queue entry
Departure of ambulance to hospital	Minor Service entry
Arrival at hospital	Major queue entry
Ambulance return to station	Major Service entry
	A&E exit

Table 4: List of events

Regional hospitals with A&E services are located in the ambulance coverage area. Ambulances are not restricted to serving a particular hospital and they can transfer an emergency patient to either nearest or most appropriate hospital. The selection of the hospital could be either on the type of services/expertise or availability. A&E function in itself is a complete process and limited to its hospital, therefore several simulation models are required to represent different A&E. Since the ambulance services will communicate with all the A&E department and make their decisions on the transfer of a patient, therefore a separate model is developed for ambulance services. This ambulance model will communicate with all other A&E models. This case study extends to apply the proposed protocol using two different distribution techniques i.e. ABS and DES. Hence, the A&E departments are modelled using DES and the ambulance service is modelled using ABS technique. Since both the simulation techniques use discrete time-stepped approach, it becomes easier to interoperate the two approaches. However, DES has two different implementations, i.e. Next Event Request (NER) or Time Advance Request (TAR). In NER, it is not necessary that there will be an event at every time tick therefore the simulation can jump to next simulation time at its next scheduled event. While in the TAR scenario the simulation will not jump to the next event, instead it will check if the event is scheduled after every single time tick. The latter strategy is easily adopted with ABS technique.

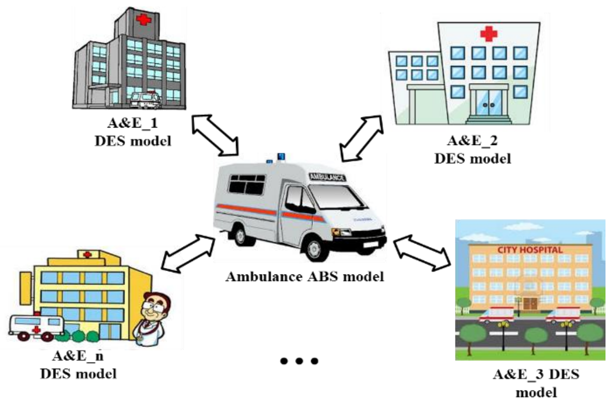


Figure 5.42: London Emergency Medical Service Model

The ambulance service model is the central component of this system and all the communication between the models will occur through this model. While, all the other A&E models will not directly communicate with each other. An abstract view of the system is presented in Figure 5.42 above, where the arrows indicate the interactions between the models.

The central Ambulance service require to find the nearest available A&E department to transfer the patient. Therefore, ambulance services need access to the availability of the A&E department status based on the type of emergency. Once an A&E is selected, the A&E department will reserve resources and prepare for the patient arrival. Hence, once an agreement to accept the patient is done, no other patient can be accepted in its place. When the ambulance arrives at the hospital, the patient object (including all its attributes) transfers to the A&E department.

The EMS case study deals with three different types of interoperability issues. It was mentioned above that the ambulance service model needs up-to-date A&E status to accept the patient. Therefore, Type D shared data structure is used to keep the information updated with the ambulance service model. Similarly, once an A&E is selected by the ambulance it will announce the emergency to A&E using Type C Shared Event. This will notify the A&E model that a patient is on its way and A&E

should reserve resources for that patient. This will also notify the Ambulance model that a particular ambulance is busy and not available to attend any casualty for next specific time until it becomes available or the patient is being transferred. The transfer of patient from ambulance to A&E is addressed by using Type A.2 (Bounded buffer) at the selected hospital. In case the hospital has no space (i.e. the queue is full), the ambulance crew might not necessarily wait instead will try to find alternative hospital.

5.4.2 Realisation of the models

All the federate models are designed to be interoperable. Therefore, the messages and information will be shared between the models are identified. A list of shared variables and its ownership in the distributed environment is drawn. The following subsections explain the implementation of each federate model and the RTI.

a) Ambulance Services model

The conceptual modelling requires the identification of the objects and their interactions. For calculating travel to the emergency and hospital a GIS grid topology was implemented. This topology was selected because of more accurate distance calculation and realistic visualisation. The ambulance coverage area is fed into this grid topology. Similarly, ambulance station locations and capacity are also fed into this grid.

The ambulance service model will have one input point, i.e. emergency call and there are two exit points. Either the patient is treated on site and then the patient is released or the patient is transferred to the selected A&E

b) A&E Model

Figure 5.41 illustrates the A&E process and the Table 4 provide the list of events. Since the focus of this case study is to test the interoperability only, therefore all the departments within A&E model are kept abstract. All the clinical resources are clubbed together. These indicate nurses, doctor and other staff. Since A&E is implemented as DES, therefore workstations and resources are hard coded. Similarly, the functioning of different departments is also coupled together and an overall three procedures are identified i.e. triage, minor and major. This model will have two input points, the first will be the walk-in patients and second will be through ambulance service. The availability of the resources, including the capacity is calculated and shared with the ambulance model when requested

5.4.3 Software Tools

For the implementation of EMS case study, it was required that simulation platform can support both ABS and DES implementation of DS techniques. Since, Repast Simphony (as described in section 4.3) support both techniques and it is also an open source tool, therefore both ambulance and A&E models are built in Repast Suite. To use this simulator for DES the fundamental components namely, work stations, resources and queues was developed and used similar to the implementation of other protocols in this thesis.

A detail discussion of HLA standard is present in various sections and it was decided to use the most recent standard i.e. HLA IEEE-1516. Therefore, similar to other implementations in this thesis PoRTIco was selected as an RTI for a communication standard for this case study.

5.4.4 Selection of Protocol

The EMS case study requires the application of three IRM's i.e. Type D, C and A.2 as described in the previous sections. Let us now examine how to select an appropriate protocol to address the interoperability issue. This will also guide the simulation practitioners (as an exemplar) on how to selected the right protocol for their problem. First, let's examine the requirements for share data structure, i.e. Type D. In this case study, the A&E in hospitals models are the owners of the information and only they have the right to update the resource status, while the ambulance model is only reading or using the information to make decisions, i.e. no updates are required for the ambulance model to this data and there is no issue for update anomaly. Also, the A&E models do not communicate with each other directly, their communication is limited to ambulance models. This means that each A&E will update its own data structure shared with only an ambulance model. Hence, a need for a separate central controller is not required. Therefore, the modular can select full replication protocol to address this interoperability issue, because the other proposed models have separate central server to handle updates anomalies.

Announcement of the emergency to all other federate or models is another type of IRM problem i.e. shared event. This event is an interrupt to notify other models about the emergency and since this data will not be required thereafter, it is therefore advisable to use the shared event protocol using interactions. However, a shared event protocol using attribute can also be used, but since a persistent memory is not required, therefore the model will not fully utilise the attribute potential.

Finally, the selection of protocol to address the interoperability issue related to transferring a patient from ambulance model to A&E model. First, consider the operation of the A&E. There are two entry points for a patient (an entity) in A&E model, i.e. walk-in-patients and ambulances. The number of resources like staff, beds and facilities are also limited. In most of the cases the number of walk-in-patient is greater than the ambulance transfers. Therefore, there will be a continuing change in the availability of resources to accept patients from the ambulance. It also seems unnecessary for the A&E model to continually update ambulance model, when this data is not required. Therefore, this process rules out the use of Bounded receiving element using Adaptive protocol because in this protocol, the receiving model/federate continually updated the other model about the status of the bounded queue. There are now two other proposed possibilities to address this issue, i.e. either using Queue Update Protocol or Bounded Buffer Update protocol. Therefore, to further filter the requirements it is required to consider the ambulance decision process for selecting the A&E. Based on the queue update protocol, if for example the simulation uses several A&E models, then ambulance model has to send messages to all models and wait for the response before making the decision for patient transfer, whilst in the case of Bounded Buffer Update, the ambulance model is kept updated with the status of A&E availability. Unlike adoptive protocol, ambulance model does not require the update on every change of the queue, instead it only needs to know if there is a space or not and this is only updated by A&E model when the queue is full or gets some space. Therefore, the information will be readily available for the ambulance model to make decisions with limited communication with other models. Hence, the bounded buffer update protocol was selected to address this approach to minimise the communication.

From the above it is possible to list the interoperability requirements and draw a template FOM for each type of federate. In summary hospital federate will send and update the resource status, receive notification of the emergency and finally it will receive the patient from ambulance model. Therefore, a parameter to represent the queue status must be defined in the FOM with read & write (i.e. Publish and Subscribe) rights and an event parameter along with patient parameter must be created with read (i.e. Subscribe) rights only. A sample FOM for one hospital is presented below.

```
<interactions>
    <interactionClass>
        <name>HLAinteractionRoot</name>
        <sharing>PublishSubscribe</sharing>
        <dimensions>NA</dimensions>
        <transportation>HLAreliable</transportation>
        <order>Receive</order>
        <interactionClass>
            <name>EntityTransfer</name>
            <sharing>PublishSubscribe</sharing>
            <transportation>HLAreliable</transportation>
            <order>TimeStamp</order>
            <parameter>
                <sharing>Publish</sharing>
                 <name>queuestatus</name>
                 <dataType>HLAinteger32BE</dataType>
```

```
      </parameter>
      <parameter>
        <sharing>Subscribe</sharing>
        <name>event</name>
        <dataType>HLAinteger32BE</dataType>
      </parameter>
      <parameter>
        <sharing>Subscribe</sharing>
        <name>patient</name>
        <dataType>HLAinteger32BE</dataType>
      </parameter>
    </interactionClass>
  </interactionClass>
</interactions>
```

Similarly, the ambulance model will receive the hospital status, send notification of emergency and transfer patients to the hospital model. Therefore, the same parameters need to be created in ambulance federate FOM but with reverse right i.e. patient and event will have write (i.e. Publish) right while queue status parameter will only have read (i.e. subscribe) right. A part from these changes, Queue status parameter must be repeated for each hospital model.

5.4.5 Simulation testing

All the models in EMS case were designed and developed as discrete-event simulation. As described in chapter 2, Discrete event simulation structured around events and processes. In an overview, an event results in some exchange of message(s) which alters the state of the system, while the processes represent a sequence of activities, this represent the core components of the system. Processes explicitly advance virtual time to represent "processing time" and implicitly advance time when events are scheduled to occur in future.

Simulation based testing is one of the testing techniques used within a simple and generic testing process (Rutherford et al., 2006). To initiate this testing, the first step was to prepare the test cases. Normally, these test cases are defined in the early stages of system development life cycle i.e. at conceptual modelling stage. Each test case will have a direct input to the system as an input vector, functional parameters, environment input and conditions. In case of EMS case study, the sequence of processes is illustrated in figure 5.41 and figure 5.42. Further the expected events are also explained in table 4.

5.4.5.1 Simulation Model Testing

There are two different models used in EMS case study one represents the Ambulance service and the other A&E department of a typical hospital. All the A&E models are identical, therefore for model verification and testing one of the A&E hospitals is used. For the purpose of testing, simulation, code, both the models were tested in this study during the development phase by using both Black box and White Box testing technique. Later this code was verified by a professional M&S programmer.

Testing for simulation execution or validation of the models, pilot runs were performed and data was collected at different point as mentioned above to ensure the models meets its requirements. Some of the data were verified from the published data to ensure the accuracy of simulation execution. For example, the 75 percent response time for life threatening (major) and 95 percent of all other emergencies are listed in LAS Annual Review (2011-12) as 8 minutes and 19 minutes respectively.

Similarly, in A&E model the time for patient journey is the key indicator. According to the published data 2011-12, all patients at A&E must be served within four hours. Therefore, the data collected at several points was checked against these indicators to verify the simulation model.

5.4.5.2 Interoperability Testing

Each hospital model run independently and they only communicate with ambulance model. Similarly, the ambulance model does not share hospital information with other hospital models. The ambulance model and all the hospital models run within a single federation and for interoperability the PoRTIco provides the basic communication, time synchronisation and message passing platform. The previous section described how the simulation models were tested and verified, but to test the distribution and the interoperability between these models' separate tests were conducted.

Therefore, the first step used was to run the single simulation execution to collect category B data. To achieve that, a number of check points were introduced to observe and collect data during the simulation executions, this include, "call generation", "ambulance dispatch", "arrival at scene", "hospitalisation required", "find hospital", "arrival at hospital", "returning back to station". A sample data collection is presented in table 5 below.

Time Tick	Call generation ID	Ambulance dispatch ID	Arrival at scene	Hospital Required	Find hospital	Arrival at hospital	Returning to station
1	1	31					
5	3	22					
9	4	11					
27	5	23					
34	1	31	33.599	TRUE	34	102.862	135.461
35	4	11	34.228	TRUE	35	60.273	85.502
37	6	12					
39	3	22	38.621	FALSE	0	0	82.621
42	7	24					
46	5	23	45.356	FALSE	0	0	84.356
49	8	13					
53	6	12	52.273	TRUE	53	83.866	99.139
57	9	32					

Table 5: Data collection of check points

Later the experiment was repeated, but this time both the models ran in a federation and similar data was captured. Although, the models are stochastic i.e. call generation, arrival times, selection of hospitals, etc. are identified at run time, therefore the runs do not produce exactly the same output. However, the data compared identified that the models are behaving the same as they were while running in a non-distributed environment.

5.4.6 Fault tolerance

When a simulation model is distributed, a crash of any federate can force the entire simulation to an end. The increase in the number of federates in a federation also increases the probability of failure during a simulation run and if the federates are not at the same geographical location, it further increases the risk. Restarting a failed federate can leave the system in an inconsistent state, therefore some sort of fault tolerance is required.

A crash of a federate is due to a problem in either software or hardware. The hardware failures can be the CPU malfunction, power cut-out or even disconnection from the network. The problems related to software can be further sub divided at different levels i.e. operating systems, RTI, simulation model and the interoperability. Operating system problem can be a process malfunction or crash of the operating system itself. On the other hand, RTI's based on HLA standard do provide fault tolerance at the communication level, i.e. if either of the federate will resigned or terminated from the federation, a simulation can continue to communicate with the remaining federates. Similarly, a federate can re-join the federation. But this does not necessarily mean that the results produced from the simulation run will not be affected. Therefore, to overcome the failure, fault tolerance strategy should be considered in the design of simulation model and the interoperability between these models.

There are many known techniques for fault tolerance in distributed simulation. These can be classified into two categories i.e. checkpointing or replication. Replication techniques have additional synchronisation overhead between its replicas, while checkpoints technique, need additional storage to save a safe checkpoint, i.e. if any failure happens the simulation could be restored to its last checkpoint. Replication techniques can be implemented through simulation model while the checkpoint technique can also be achieved by using the standard RTI funtions. But both of them are only possible if the simulation does not crash at the time of failure. The reasons for a simulation crash can be a deadlock i.e. if a federate is expecting a response from another federate which have failed and will not respond. In such circumstances the simulation can not be restored using any strategy, therefore it is important that the interoperability protocol used between the federates also support fault tolerance.

The proposed protocols in the given DSI framework are designed to support such conditions. For example, in the case study of London emergency medical services, there are several A&E federates communicating with the ambulance model. The entire simulation may halt if any one of the A&E models fail. But the interoperability protocols are designed to avoid deadlocks conditions i.e. when A&E model fail to respond its availability, no value will be returned which, by default will be assumed as non-available and the ambulance model will search another available hospital to transfer the patient. It is very important to note that recovery from fault tolerance is only possible if the simulation is still in a running state since interoperability protocols are running at the grass root level in the distributed simulation, any deadlock at this level could result in simulation halt.

5.5 Summary

The first sections of this chapter addressed the need for independently developed distributed models representing generic interoperability problem scenarios for running experiments. The use of software tools for developing the models was then followed by a reasoning for selecting different versions of HLA and PoRTIco for development.

The remaining sections of this chapter focused on the conceptual modelling of optional proposed protocols to address each IRM issue.

A graphic representation of the relationship between federate models, the simulation package, and the RTI was included for each principle IRM Type. Two basic methods of entity transfer were established: using interaction class with attribute parameters, or using object class via publish/subscribe with attributes. The FOM used to define the interaction class was set out for each problem case and the sequence of events for implementation was described, step-by-step, for all proposed case protocols for every IRM Type.

Each proposed protocol with optional case scenarios was presented independently to ensure clarity of procedure for practitioners. The next chapter presents an evaluation of experimental results using the DSI Framework.

Chapter 6: Evaluation of the Framework

6.1 Overview

Previous chapters discussed conceptual models and the design of different approaches to addressing the IRMs. In total, seventeen different approaches were discussed and proposed. Some of these approaches are based on different case studies to evaluate performance under different conditions.

The focus of this chapter is on experimental runs and the analysis and evaluation of data collected from them. The chapter, first describes the simulation test environment, i.e., network, software, and equipment. This is followed by presentation and discussion of the data collected from implementation with graphical representation and some comparisons. The examination of results presented in this chapter concludes the evaluation of the hypothesis. Finally, the proposed DSI Framework is revisited, together with some further recommendations.

6.2 Testing the simulation environment

To test the hypothesis, the DSI framework presented a number of solutions for each interoperability issue presented in IRM. These solutions are not exhaustive, but can provide options to select an appropriate solution for the distributed simulation practitioner's problem(s) or point the practitioner in the right direction. These solutions are kept purposely generic for ease of understanding and adoption for any scenario because (as discussed in previous chapters) a practitioner will not necessarily require all the given interoperability issue solutions. Similarly, the OMT and the sequence of events are also explained separately, to help the practitioner avoid unnecessary implementation. The different solutions presented purposely use similar variable names to assist in better understanding. The variable names used are also generic, and can be adopted by the practitioner according to their simulation models. Some of the proposed protocols were run using different distribution values to experience the impact on system behaviour of different model environments.

6.2.1 Network Setting

A dedicated LAN network was established for running the proposed simulation models with PCs connected over 100Mbps. To maintain consistency, all the models were executed on the same PCs and in the same environment. An average of five runs for each protocol was recorded to reduce variance due to operating system and network communication (Mustafee et al., 2009; Taylor et al., 2002b). The network consisted of three PCs interconnected via LAN and configured in a star topology. Each PC was branded Dell Optiplex 745 with Intel(R) Core(TM) 2 Duo processor E6400 2.13Ghz, 4.00GB RAM and 80GB secondary storage. Each PC had a fresh Microsoft Windows 7 (64 bit) operating system installation to avoid unnecessary background application delays. To achieve best network performance no additional Internet security or firewall was installed. The software installed included Java 1.7 JRE with Repast Simphony 2.1; poRTIco 2.1.0 package with HLA 1516e; and HLA 1.3 RTI support.

6.3 Performance Testing

Usually in a typical simulation V&V testing, some data are collected to compare with the results of test runs for analysis and evaluation, but in this case category C data were used, as described in Chapter 4, because the protocol is generic. Performance testing is not just a simple task of comparing test results. First, we need to decide the parameters of performance measurement, only then can the performance be benchmarked. Defining measures is important because in a complex system there are different possible setups that a practitioner might use, and performance parameters might also be different. In this case, the performance is measured for a distributed simulation where we have already established that because of complexity and difficulty such simulations cannot be executed on a single machine. They are therefore distributed over a network of CPUs arranged in any order. This means the performance testing that was conducted completely omits performance comparison of a simulation running between a single node and a set of networked nodes. However, such a study was conducted in work previously published by Nouman et al. (2013) and discussed later. The simple solution to single node overload is to distribute the load to multiple nodes, but the real problems come when these nodes interoperate with each other. The interoperability solution has already been covered in this research, but now the question is about performance.

Network communication is considered a bottleneck and great effort is placed on reducing this bottleneck through considerable research, as the speed of execution of a CPU is much faster than network communication speed. Additionally, the introduction of network security, such as the firewall, is contributing to further reducing speed. Network communication plays an important role in distributed simulation performance, and therefore the focus of performance in a distributed simulation is on the measurement of time. A single microsecond delay in a single simulation time tick will make a huge difference when the same simulation is run to monitor effects for a month, a year, or more.

This research uses an RTI based on the IEEE 1516-2010 standard at its heart of network communication. Therefore, communication depends on federation configuration and implementation such as in time synchronisation. For example, over time, synchronisation requires the federate to have to block and wait for approval from RTI to advance time. Again, the main focus of this research is not to measure the performance of the RTI, but some results are compared with different RTIs to monitor the possible impact of using different products. Every RTI vendor tries to implement the best possible solution to improve the performance of their product. For example, MAK RTI optimise data transfer by placing the computers in the role of HLA switches/routers. Similarly, pitch RTI uses a booster application to speed up communication by simplifying network and firewall configuration issues. Hence, the main focus of this research is to measure the performance of each of the IRM protocols explained in the last chapter. With some of the initial research, it was concluded that the following parameters can have a significant impact on performance:

- The model design
- The selected RTI
- The operating system configuration
- The selected hardware configuration
- The network configuration

The performance statistics (discussed later) will help practitioners determine the impact of implementing strategies to deal with the interoperability issues. The test run results for each IRM are discussed individually, as explained in Section 5.2. The research uses speed as a matrix to measure the performance because distribution is required to improving the execution speed of a simulation (Fujimoto, 1987). Similar matrics were used in the most recent research by Fujimoto (2015), Feldkamp et al., (2015) and Anagnostou and Taylor (2017).

6.3.1 General Entity Transfer

Four different protocols explained in Chapter 5 (Section 5.2.2) for General Entity Transfer were executed five times each to capture the performance parameters. The performance results for these protocols are shown in Table 6. The test was conducted for up to 5 weeks of simulation run time and the average (Avg.) run of each protocol is shown in this table. Also, the table lists the standard deviation (SD) of five runs for each protocol. The standard deviation is within acceptable limits.

Simulation Run time	**1 Week**		**2 Weeks**		**3 Weeks**		**4 Weeks**		**5 Weeks**	
Protocol / Execution time	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**
Interaction	15.45	0.26	30.01	0.54	44.69	0.63	59.20	0.80	74.19	0.76
Interaction with Null msg	15.38	0.21	31.05	0.43	45.97	0.70	60.77	0.82	75.81	0.78
Attribute	15.23	0.55	29.88	0.18	44.44	0.83	59.12	0.61	73.31	0.61
Attribute with Null msg	15.20	0.53	30.52	0.61	44.79	0.77	60.15	0.60	75.13	0.49

Table 6: General Entity Transfer Performance

The general entity transfer interoperability problem is generic and common in distributed simulations. The solution is achieved by message passing via RTI. There are two possible ways for message passing, i.e., either by interaction or by attributes. The RTI implementation of interactions is defined as non-persistent memory, while that for attributes is defined as persistent memory. The above table can be seen as comparing the difference between use of interaction classes and attribute classes and then a comparison between two different approaches: i) a simple approach with continuous update using null message technique, and ii) on demand.

The graphical representation of the performance results are shown in Figure 6.1, which presents the combined test run results for all four protocols. Figure 6.2 illustrates comparison figures in a pair of different combinations. The Y-axis on these graphs shows the execution time in minutes while the X-axis on these graphs shows the number of weeks.

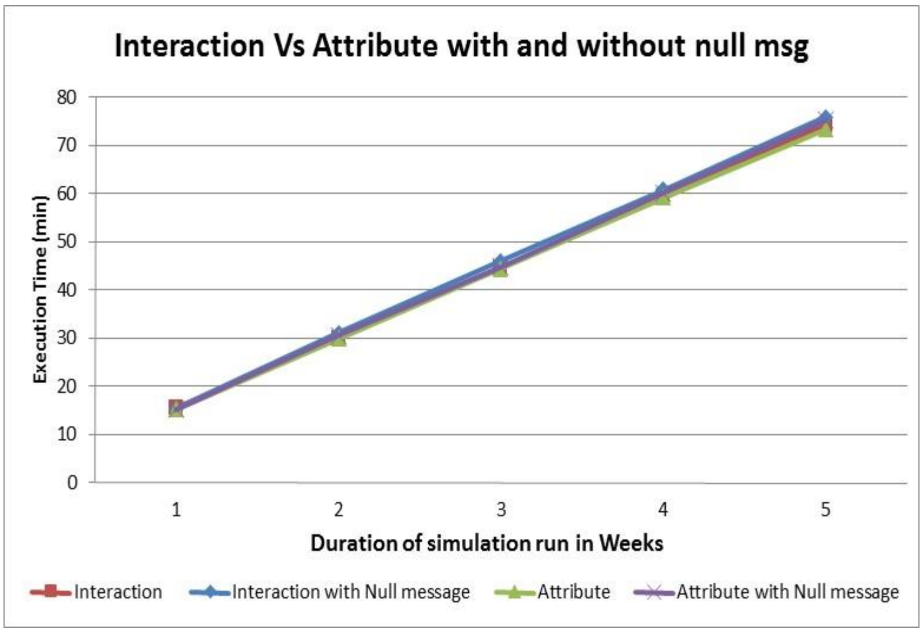


Figure 6.1: Execution time for General Entity Transfer proposed protocols

The graphical representation of the test run illustrates little difference, and the results look mostly similar. But a close examination of data presented in Table 6 indicates slight differences between these approaches. Although the difference is not significant in generic model implementation, but it still exists. Figure 6.2(a) presents a comparison between two different approaches using interaction classes. Similarly, Figure 6.2(b) illustrates the comparison between two different approaches using attribute classes. Finally, Figure 6.2(c) illustrates the difference between use of interaction class and attribute class.

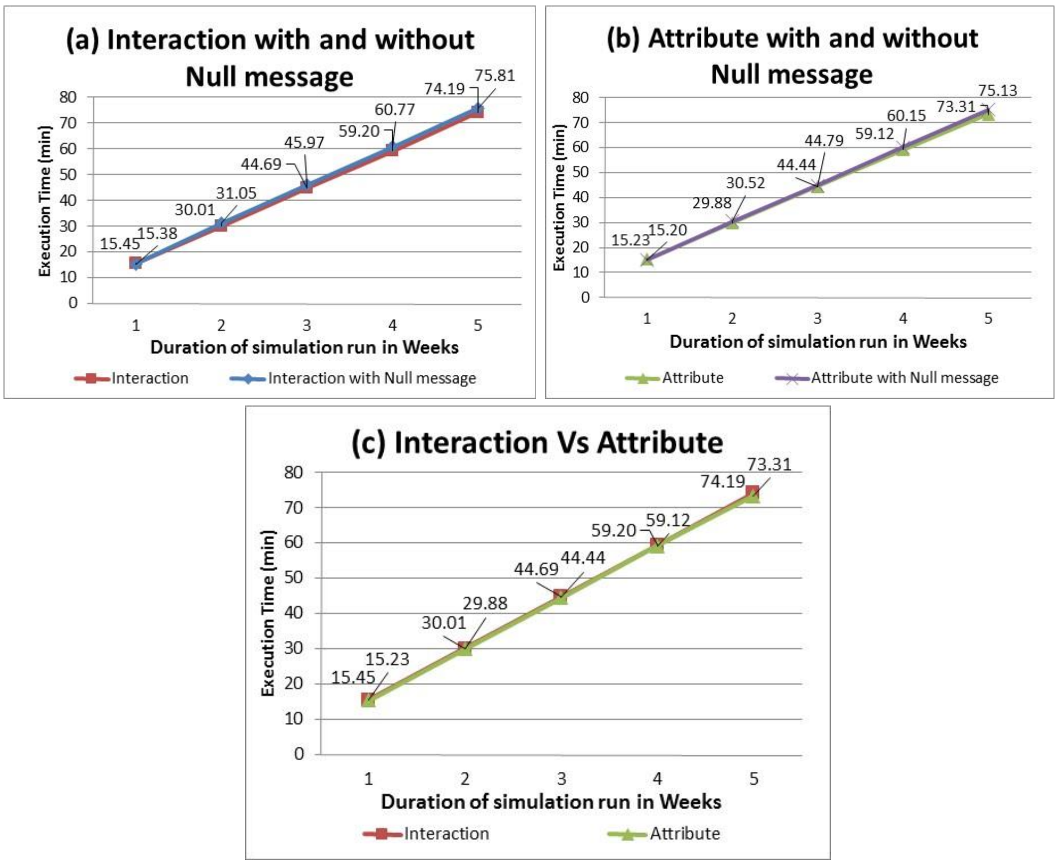


Figure 6.2: Comparison for General Entity Transfer proposed protocols

As can be seen, as the simulation run time increases the difference in null message technique also increases. The difference is not very prominent because all the experiments were run in a dedicated network environment and the main objective of these experimental runs was to demonstrate the solution for the IRM Type A.1 problem and identify the performance of two possible approaches for message passing. Further, the impact of dedicated network and non-dedicated network are explored in experimental results in the bounded receiving element approach (next section). As expected, the difference between the use of interaction and attribute is hardly visible and in some execution runs they overlap. Therefore, the experimental runs shown above demonstrate that the two approaches have no or negligible overhead over each other and the choice does not compromise performance. This is unsurprising considering the issue is simple entity passing and no significant action is required by any model upon receipt of the message.

6.3.2 Bounded Receiving Element

Three different protocols were executed with different scenarios, RTI, and environment to capture the performance parameters. Since it is being concluded from our previous experimentation that use of either attributes or interaction class has no significant effect, the research continues by using interaction class for message passing for the rest of the implementation. Further details for selecting the interaction classes is discussed in next chapter. The three protocols were explained in more detail in Chapter 5 (Section 5.2.3), but the three different scenarios used for each protocol execution are based on the state of a bounded queue and are listed as:

1) Unlimited Queue
2) Partially Full
3) Always Full

In an unlimited queue scenario, the bounded buffer never blocks. This means the entity will be transferred without blocking the sender federate. But in the partially full scenario, the bounded buffer behaviour will change over a period of time, i.e., at unpredictable times the buffer will be full and block the incoming entity. In the always full scenario, the bounded buffer will remain full at all times and will block incoming entities. The results of these different protocols and scenarios are illustrated in Table 7(a)(b) and (c). The experimental runs were also conducted in a dedicated environment.

The test was conducted for up to 5 weeks of simulation run time and an average (Avg.) run of each protocol is shown in table 7. The standard deviation for all runs conducted in a dedicated networked environment were similar to the previous results, i.e., below 1. Therefore, the standard deviation is not included in the tables below. The graphical representation of the data is illustrated in Figure 6.3(a)(b) and (c). The Y-axis on these graphs shows the execution time in minutes while the X-axis on these graphs shows the number of weeks.

(a) Unlimited Queue										
Simulation Run time	**1 Week**		**2 Week**		**3 Week**		**4 Week**		**5 Week**	
Protocols	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**
Queue update	16.24	0.36	32.75	0.31	48.15	0.42	63.01	0.52	80.30	0.54
Buffer update	15.27	0.57	30.28	0.42	45.06	0.28	59.19	0.39	77.42	0.67
Adaptive	15.14	0.41	29.98	0.63	44.29	0.32	58.99	0.57	76.15	0.51

(b) Partially Full										
Simulation Run time	**1 Week**		**2 Week**		**3 Week**		**4 Week**		**5 Week**	
Protocols	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**
Queue update	21.31	0.68	42.28	0.72	63.12	0.69	84.83	0.75	106.25	0.87
Buffer update	18.11	0.81	35.75	0.83	53.45	0.78	71.46	0.77	89.51	0.83
Adaptive	15.23	0.76	30.46	0.74	45.26	0.81	60.99	0.75	75.95	0.79

(c) Always full										
Simulation Run time	**1 Week**		**2 Week**		**3 Week**		**4 Week**		**5 Week**	
Protocols	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**
Queue update	15.52	0.57	30.67	0.48	45.83	0.34	61.21	0.74	76.13	0.64
Buffer update	15.40	0.53	30.69	0.47	45.60	0.51	61.16	0.36	76.02	0.32
Adaptive	15.23	0.59	30.31	0.55	45.38	0.54	60.58	0.41	75.81	0.48

Table 7: Bounded Receiving Element PoRTIco Dedicated Network

Three different scenarios (i.e. unlimited queue, partially full and always full) were applied to three different protocols (i.e. Queue update, Buffer update and Adaptive) to monitor the impact of each protocol performance on the change of scenario. It can be noted from the figures and graphs that Unlimited Queue and Always Full Queue have some similarity. But in Unlimited Queue the queue update performance is less than the other protocols because it still sends extra messages to update queue to the sending federate.

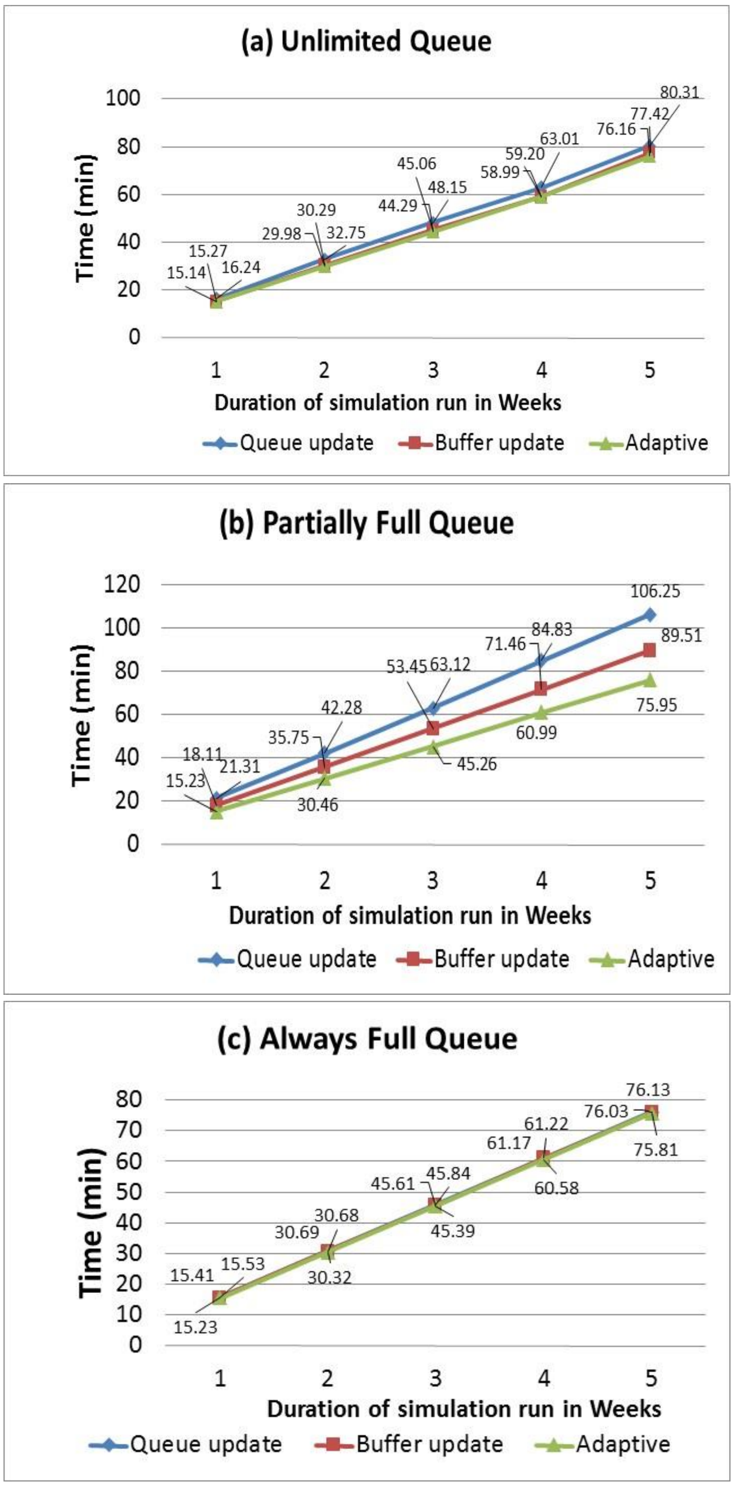


Figure 6.3: Comparison for Bounded Receiving Element using PoRTIco on a Dedicated Network

In Always Full scenario, all three federates request the same information and also process in similar fashion because of the blocked nature of the receiving queue. In Partial Full scenario, each protocol has a different run time because of the multiple request and block procedure. As expected, the adaptive protocol outperforms the others.

The best two protocols based on performance were selected for further experimental runs on a non-dedicated network to experience the impact of applying these protocols on a shared network. Table 8 illustrates the averages for a five week run on a non-dedicated network. The standard deviation for all runs conducted in a non-dedicated networked environment was higher than the dedicated environment, ranging between 2 and 4.

(a) Unlimited Queue										
Simulation Run time	**1 Week**		**2 Week**		**3 Week**		**4 Week**		**5 Week**	
Protocols	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**
Buffer update	17.97	1.45	36.24	1.68	54.31	1.52	71.91	1.21	88.42	1.87
Adaptive	17.50	1.59	34.73	1.87	51.65	1.24	69.70	1.73	86.15	1.64

(b) Partially Full										
Simulation Run time	**1 Week**		**2 Week**		**3 Week**		**4 Week**		**5 Week**	
Protocols	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**
Buffer update	17.97	2.14	36.24	2.76	54.31	3.14	71.91	2.25	88.42	2.16
Adaptive	17.50	2.37	34.73	2.15	51.65	1.98	69.70	2.24	86.15	2.37

(c) Always Full										
Simulation Run time	**1 Week**		**2 Week**		**3 Week**		**4 Week**		**5 Week**	
Protocols	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**
Buffer update	17.97	2.58	36.24	2.81	54.31	2.14	71.91	3.54	88.42	2.85
Adaptive	17.50	2.75	34.73	1.85	51.65	2.49	69.70	2.82	86.15	1.65

Table 8: Bounded Receiving Element PoRTIco Non-Dedicated Network

The graphical representation of the data presented in Table 8(a)(b) and (c) is demonstrated in Figure 6.4(a)(b) and (c). As expected, the behaviour of these protocols is similar on both dedicated and non-dedicated networks with only one exception, i.e., the total execution time. These protocols run faster over a dedicated network compared to a non-dedicated network as illustrated in Figure 6.4(d). The reason for the longer execution time is not known, but it is almost impossible to calculate the performance, delay of all the variables, i.e., type of network equipment, network topology, firewall, background process, etc. Although the same PC's were used for this experiment they were connected over the Brunel University network. This experiment indicates that running the same protocol will show different performance over different networks.

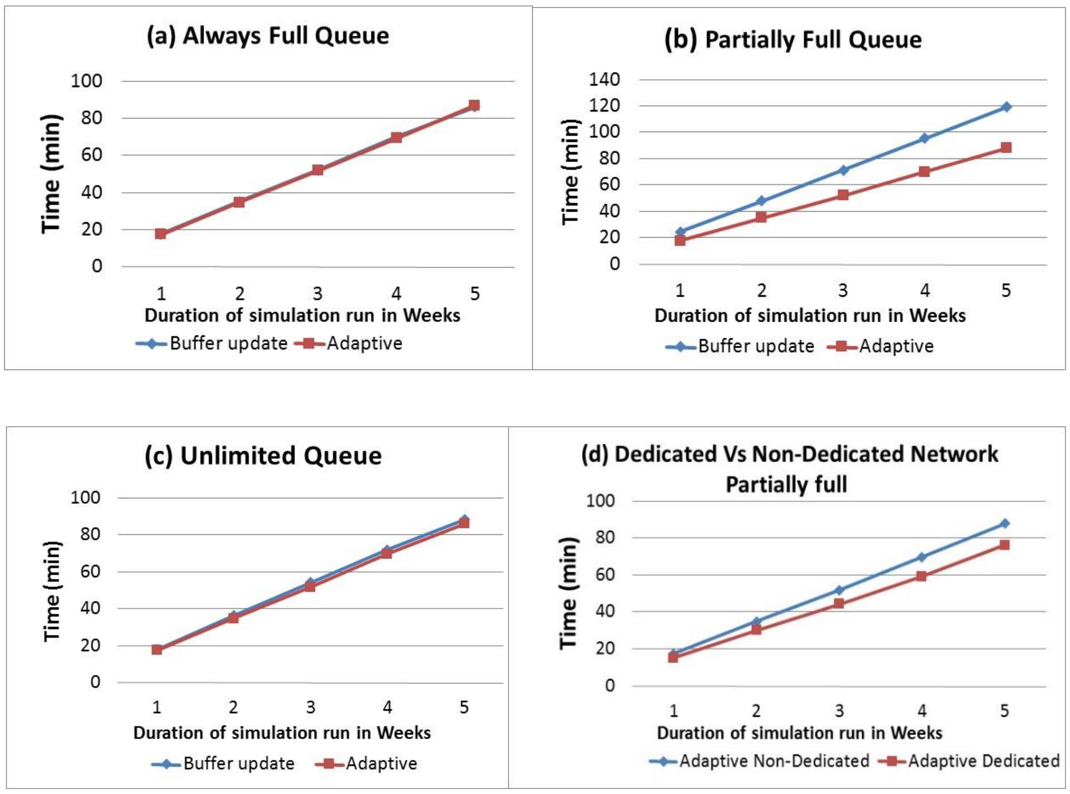


Figure 6.4: Comparison for Bounded Receiving Element using PoRTIco on a non-dedicated Network

Both of these protocols were further tested in a non-dedicated network environment with a different commercial RTI, i.e., Pitch. The intent of this experiment was to establish if different RTI has similar performance or they differ in implementation, as per the studies. It cannot be established from this research that the performance of a particular RTI will not change in its subsequent versions, but the research gives an indication of the effect of different implementations. Table 9 below shows the results of these experimental runs.

(a) Unlimited Queue										
Simulation Run time	**1 Week**		**2 Week**		**3 Week**		**4 Week**		**5 Week**	
Protocols	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**
Buffer update	39.52	2.25	69.811	1.87	104.83	2.95	138.48	2.18	175.16	2.38
Adaptive	32.47	2.75	64.99	2.14	97.70	2.24	129.42	2.24	162.54	1.81

(b) Partially Full										
Simulation Run time	**1 Week**		**2 Week**		**3 Week**		**4 Week**		**5 Week**	
Protocols	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**
Buffer update	38.81	3.14	80.31	2.89	121.08	3.35	163.18	2.16	204.60	2.47
Adaptive	32.48	3.59	65.08	3.27	97.37	2.76	130.17	3.18	162.40	2.51

(c) Always Full										
Simulation Run time	**1 Week**		**2 Week**		**3 Week**		**4 Week**		**5 Week**	
Protocols	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**
Buffer update	65.49	2.18	131.86	2.86	196.76	2.26	264.24	1.29	331.12	2.74
Adaptive	32.65	2.53	64.94	1.54	97.61	2.84	129.72	2.12	161.41	2.59

Table 9: Bounded Receiving Element Pitch Non-Dedicated Network

Figure 6.5(a)(b) and (c) demonstrate the behaviour of the two selected protocols using three different scenarios. The behaviour of these protocols in Unlimited Queue and Partially Full scenarios is similar to the above, but the behaviour of these protocols in Always Full Queue is unexpectedly not the same. It is a known fact that most commercial RTI's wrap the core RTI standard implementation under additional features and it is speculated that these wrappers might have an effect on performance. This can also be demonstrated from Figure 6.5(d) where PoRTIco outperforms Pitch RTI and the difference increases as the simulation run time increases. This does prove that the selection of an RTI will have some performance effects.

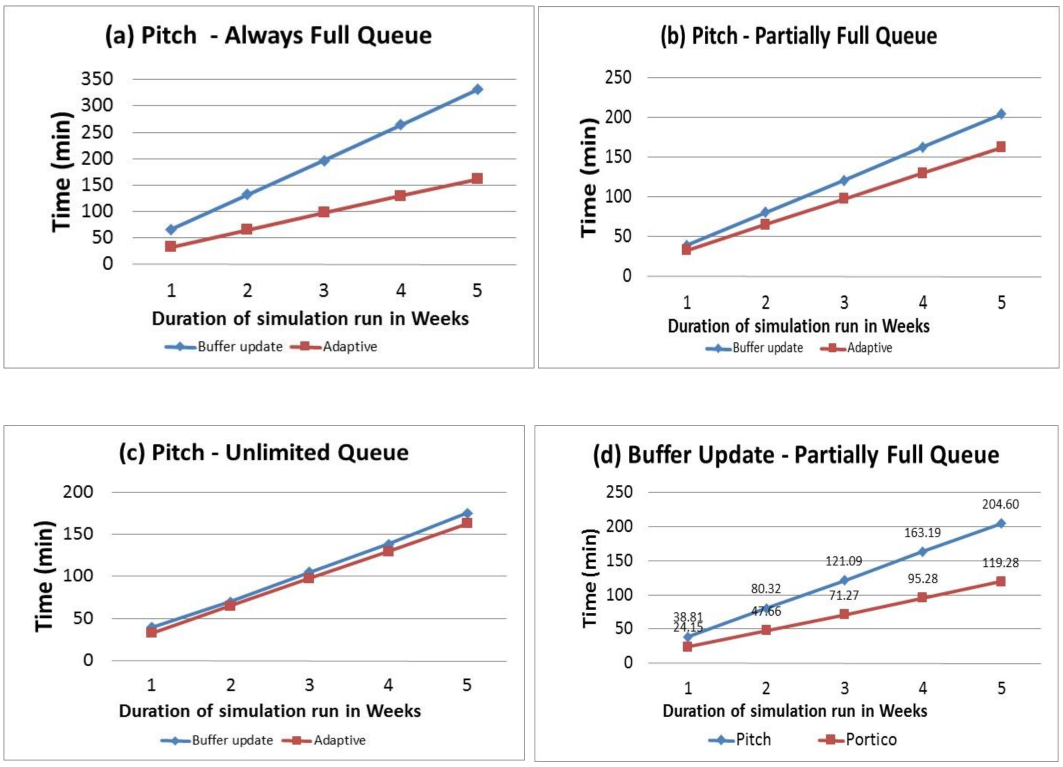


Figure 6.5: Comparison for Bounded Receiving Element using Pitch on a non-dedicated Network

6.3.3 Multiple Input Prioritisation

For multiple input prioritisation two different protocols were discussed in Chapter 5 (Section 5.2.4). The experiment strategy was the same as used in the previous sections, but with no further testing in a non-dedicated environment. All further experiments were conducted in a dedicated environment and executed five times each to capture the performance parameters. The results of the two protocols are shown in Table 10. Similarly, the tests were conducted for up to 5 weeks of simulation run time and an average (Avg.) run of each protocol is shown in this table. The standard deviation of five runs is below one and is not included in the table.

Simulation Run	**1 Week**		**2 Week**		**3 Week**		**4 Week**		**5 Week**	
Protocols	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**
Dynamic	15.19	0.24	30.10	0.52	45.15	0.27	60.15	0.43	75.13	0.26
Specialised	14.92	0.18	30.07	0.29	45.04	0.14	60.03	0.29	75.08	0.41

Table 10: Multiple Input Prioritisation Performance

The above table and the graphical representation of the data in Figure 6.6, express very little difference between the two protocols. The graphical representation follows the same pattern, i.e., execution time in minutes is shown on the Y-axis while simulation run duration in weeks is shown on the X-axis. The difference is only visible on close examination of the figures listed in the table. The reason for specialised protocol performing slightly better is because the priority list is static and does not need to generate a new list when it receives entities simultaneously. Also note that simultaneous entities are generated purposely at a more random time to demonstrate the behaviour of multiple input prioritisation.

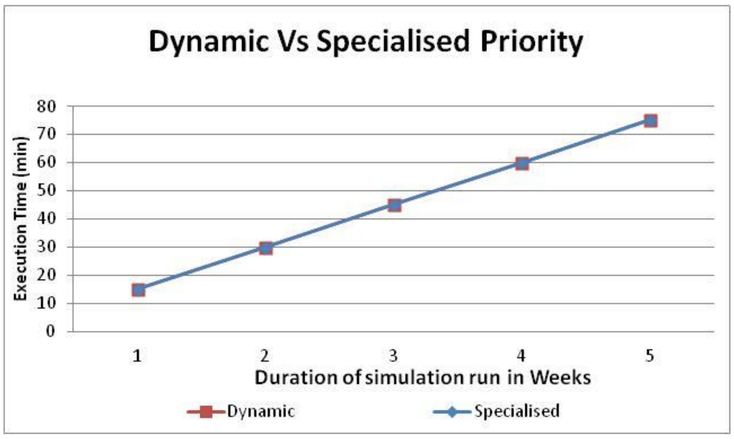


Figure 6.6: Execution time for Multiple Input Prioritisation proposed protocols

The advantages of dynamic and specialised have been discussed briefly in Chapter 5 (Section 5.2.4) and based on advantages and disadvantages of these protocols and the data from the test results, it would be hard to say that any of these protocols is better than the other. In fact, a selection can be made based on the simulation model requirement.

6.3.4 General Shared Resource

To address general shared resource issues in distributed simulation four different approaches were discussed in Chapter 5 (Section 5.2.5). In this section, two of these approaches were experimented with using different distributions to access the behaviour in different circumstances, similar to the tests conducted for the bounded buffer problem. The next request protocol was tested with three different distributions for service time named Test 1, Test 2, and Test 3 and each test given different service time distributions of (μ=4, σ = 1.5), (μ=4, σ = 1.7) and (μ=5, σ = 1.7), respectively. The change in distribution alters the resource request time, i.e., in Test 3 there will be less resource request while in Test 1 the resource request will be higher. These different tests were scheduled to identify if the change in distribution would affect the performance as expected.

Similarly, next resource event was tested with two different distributions for service time, named Test 1 and Test 2, with different service time distributions of (μ=4, σ = 1.5) and (μ=5, σ = 1.7), respectively. Again, the idea is to identify change in performance with change of distribution. The experiment strategy is the same as used in the previous sections. All experiments are conducted in a dedicated environment and executed five times each to capture the performance parameters. The results of all the protocols are presented in Table 11. As previously, the tests were conducted for up to 5 weeks of simulation run time and an average (Avg.) run of each protocol is shown in this table. The standard deviation of five runs is below one and is not included in the table below.

The graphical representation of the performance results is shown in Figure 6.7. This graph presents the combined test run results for all four protocols while Figure 6.8 illustrates comparison figures with different combinations of protocols and test cases. The Y-axis on these graphs shows the execution time in minutes while the X-axis shows the number of weeks. Figure 6.8(a) presents a comparison between the three different test cases for next request protocol as described above, and Figure 6.8(b) illustrates the comparison between the two different test cases for next resource event protocol, while Figure 6.8(c) presents a comparison between next resource, message queue, and next resource event protocols.

Simulation Run / Protocols	1 Week		2 Week		3 Week		4 Week		5 Week	
	Avg.	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**
Resource Update	30.3	0.67	60.23	0.74	90.35	0.63	115.75	0.58	148.22	0.82
Next Request - Test 1	15.63	0.57	30.29	0.58	44.73	0.46	59.72	0.61	72.51	0.47
Next Request - Test 2	14.64	0.45	30.06	0.37	45.97	0.62	60.72	0.53	75.85	0.42
Next Request - Test 3	14.81	0.54	29.69	0.41	44.16	0.58	60.08	0.37	74.12	0.56
Message queue	15.19	0.78	30.24	0.67	45.01	0.58	59.45	0.73	73.9	0.68
Next Resource Event - Test 1	15.97	0.64	29.72	0.55	44.81	0.79	61.03	0.48	75.76	0.53
Next Resource Event - Test 2	15.05	0.83	29.64	0.85	44.25	0.72	60.28	0.62	74.27	0.73

Table 11: General Shared Resource Performance

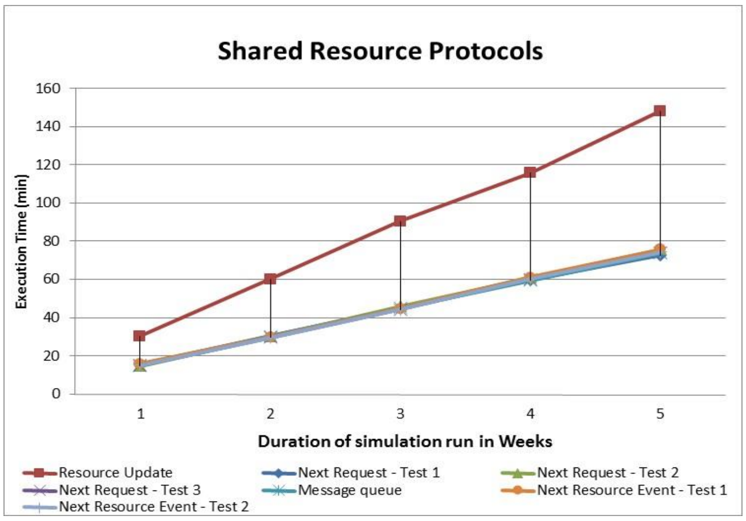


Figure 6.7: Execution time for General Shared Resource protocols

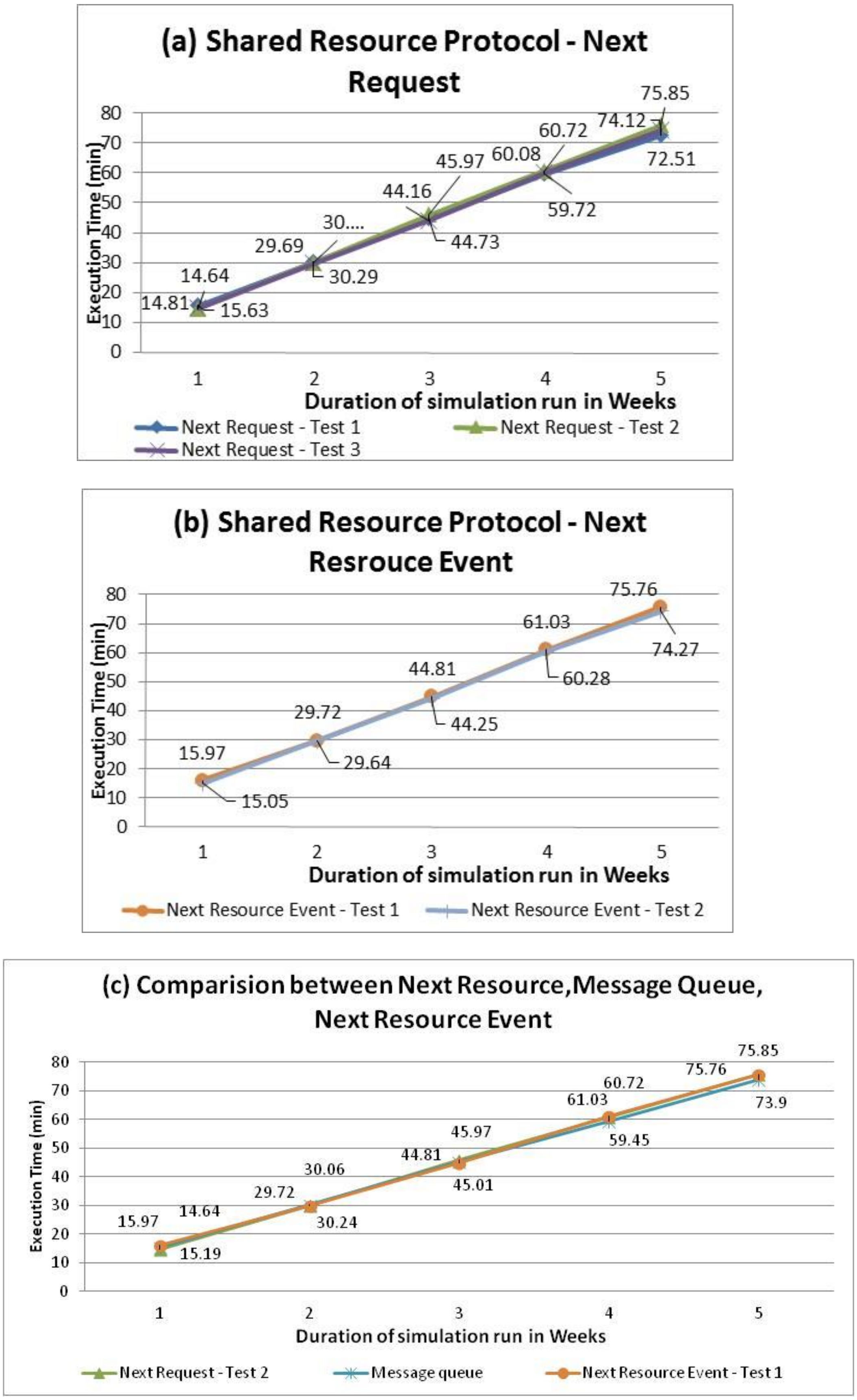


Figure 6.8: Comparison of different shared resource protocols

Figure 6.8(a), the comparison of different test cases for the next request protocol, identifies the effect of different distributions on this protocol as expected. After close examination, we can see that increase in the standard deviation in the normal distribution increased the execution time in Test 2, while increase in mean reduced the execution time demonstrated in Test 3.

Similarly, Figure 6.8(b), compared the test results for next resource event and similar behaviour is observed, i.e., having greater mean for service time increased the performance of the protocol. Reducing and increasing the value of mean in normal distribution of service time means fewer resource requests. But in either case, the difference is not phenomenal and is still in acceptable range.

From Figure 6.7 and the data presented for resource update protocol, it is clear that performance is not so good. There is performance drawback, despite this protocol being less complex and easier to implement. Therefore, a comparison between the three protocols is illustrated in Figure 6.8(c). The test results indicate that these three protocol have similar performance but they have some limitations, i.e., next resource protocol does not guarantee resource time while message queue provides a time guarantee. Similarly, next resource event can only be used with distributed simulation using the even list approach, and it is not suitable for other types of simulation. Message queue is complex to implement, but is more scalable, and as the simulation progresses further its performance increases.

6.3.5 General Shared Event

General shared event, as explained in Chapter 5 (Section 5.2.6), is more about the technique and process following the trigger of a shared event. Shared events can be triggered by a simple message similar to entity passing; the only difference is the occurrence of event. Two test scenarios were created, one ran a broadcasted event using interaction classes, and the other involved an event determined by attribute. As discussed earlier, these are the only two methods provided by RTI to transmit messages between participating federates in a federation. The results of these experimental runs are shown in Table 12. The test was conducted for up to 5 weeks of simulation run time and the average (Avg.) run of each protocol is shown in the table. The standard deviation is within acceptable limits. As explained in Chapter 5, the events are generated by using normal distribution of mean, $\mu=100$, and standard deviation, $\sigma = 15$.

Simulation Run	1 Week	2 Week	3 Week	4 Week	5 Week
Interaction	14.87	29.92	44.96	59.19	74.19
Attribute	14.73	29.88	44.44	59.09	73.21

Table 12: General Shared Event Performance

A graphical representation of the performance results is shown in Figure 6.9. This graph presents the combined test run results in both the cases. The Y-axis on these graphs shows the execution time in minutes while the X-axis shows the number of weeks.

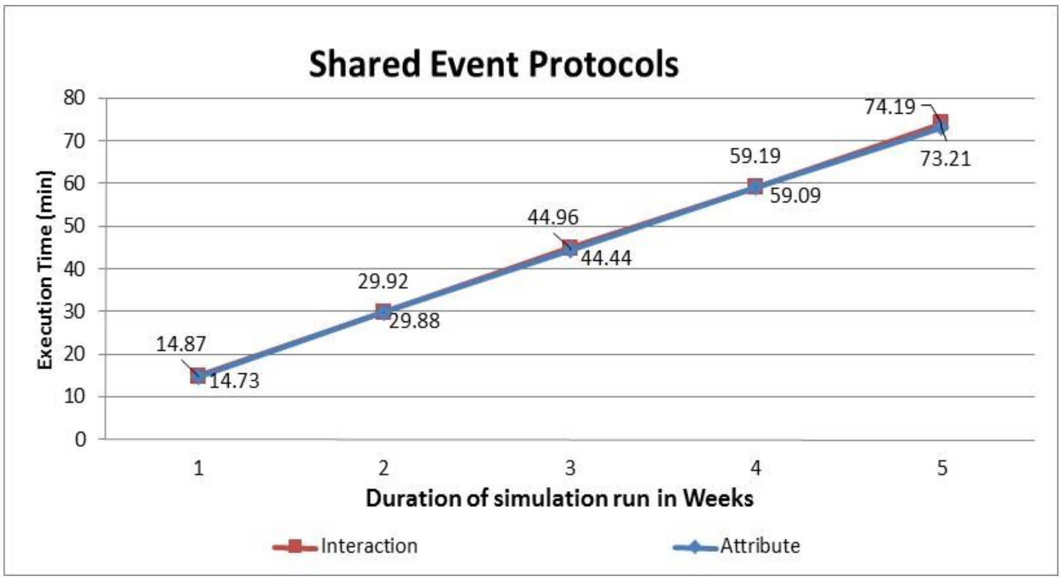


Figure 6.9 : Execution time for General Shared Event protocols

We can observe that the difference in execution follows a similar trend to the test results received for general entity passing. The one noticeable difference is the reduced overall execution time for both experimental runs as compared to general entity passing. This is because the implementation does not include any entity passing and both the models are independent. They are only time synchronised with each other and pass a message when an event occurs. Hence the models have limited interaction with each other.

6.3.6 Shared Data Structure

Three different approaches were implemented to resolve shared data structure problems. Two of these approaches, i.e., central control and central update, required a minimum of three nodes, while the full replication experiment runs were conducted on two nodes. As discussed in Chapter 5 (Section 5.2.7), both central control and central update could also be implemented using two nodes where one of the nodes would have to perform a double role, and the balance of load distribution would not be the same. Therefore, it was decided to use three nodes for the implementation and experiment runs. Also, having a separate central node supports scalability. The results of all the protocols are shown in Table 13. As before, the tests were conducted for up to 5 weeks of simulation run time and the average (Avg.) run of each protocol is shown in the table. The standard deviation of five runs is below one, and therefore not included in the table.

Simulation Run	**1 Week**	**2 Week**	**3 Week**	**4 Week**	**5 Week**
Central Control	15.03	30.82	45.41	60.64	75.93
Central Update	15.05	30.06	45.05	59.88	75.04
Full Replication	29.33	58.22	89.56	117.87	141.99

Table 13: Shared Data Structure Performance

The graphical representation of the performance results is shown in Figure 6.10. This graph presents the combined test run results for all three protocols. The Y-axis on these graphs shows the execution time in minutes while the X-axis shows the number of weeks. The difference in execution time for full replication and the other two protocols can be seen quite noticeably. This is because of the extra synchronisation required to provide data consistency. In shared resource, the challenge is limited to a read and insert anomaly, while in the shared data structure an additional challenge of update anomaly needs addressing. This feature is not known to be fully supported by the RTI. Using an additional database server can also resolve this issue, but this is outside the scope of this research and was not considered for testing.

The comparison of test results for central control and central update does not reveal major differences, but again there are some limitations, which can affect simulation performance given sizable execution run time. The performance of central update is slightly better than central control because with central control the federates have to communicate more frequently with the central node to read data value and to send the value as an update. In central update, a copy of the data is kept locally for read operation and no federate has to communicate with the central node to read the values, except when they need to update the value. The difference in both central update and central control will vary based on the frequency of updates. If the updates rarely happen, then central update can easily outperform the central control protocol.

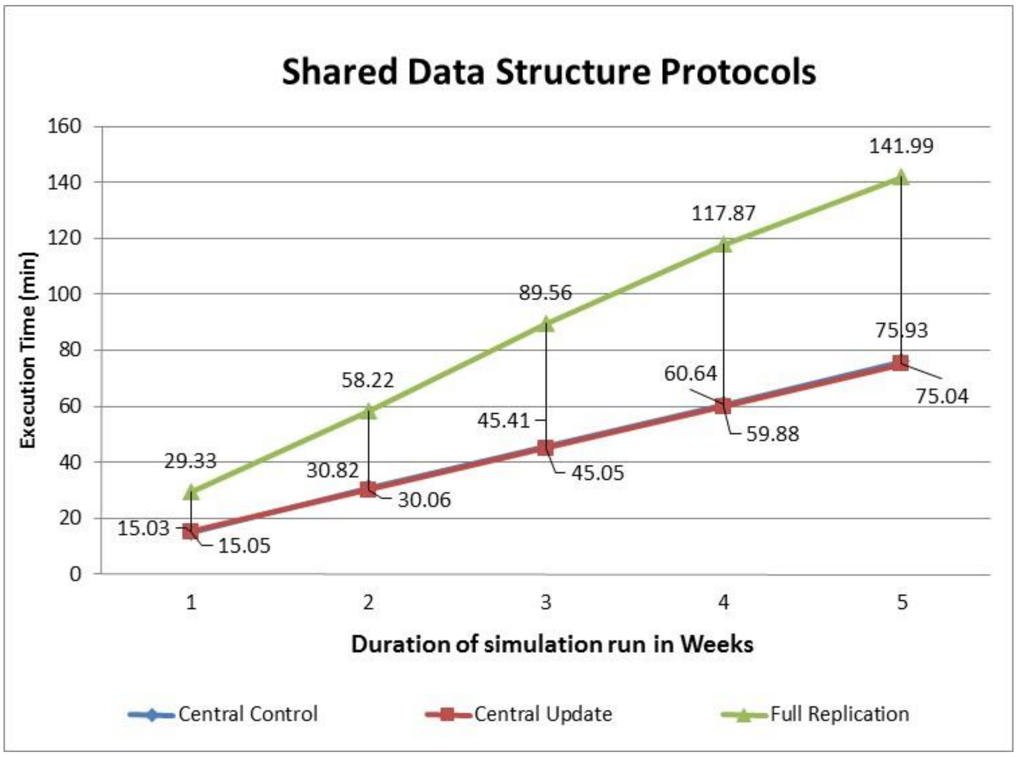


Figure 6.10: Execution time for shared data structure protocols

6.4 EMS Model testing results

In chapter 5 London EMS case study was introduced to demonstrate how the proposed protocols could be applied on a practical scenario. To evaluate these protocol two different types of experiments were conducted i.e. Performance testing and Scalability testing. For performance testing the A&E and Hospital models were executed number of times by increasing the duration of the simulation run while for scalability testing the number of federates were increased to observe the effect on simulation. The results are shown in table 14 with their Standard deviation (SD). The experiment runs were performed under the same network settings as mentioned in section 6.2.1.

No. Federates / Run Time	2 Weeks		4 Weeks		6 Weeks		8 Weeks	
	Min	SD	Min	SD	Min	SD	Min	SD
2	136.3	0.81	665.7	2.43	1405.7	1.26	2843.1	2.35
4	144.6	1.25	675.2	0.93	1412.0	0.78	2851.0	1.76
6	149.7	2.46	685.4	1.35	1418.2	0.89	2857.3	0.87
8	158.0	1.65	694.3	2.32	1424.2	2.34	2862.0	1.23
10	166.3	2.40	703.0	1.45	1430.1	1.69	2868.9	0.84

Table 14: EMS Distributed Network run

6.4.1 Performance Testing

To measure the performance of the protocol it is necessary to analyse the data in order to note predictable behaviour or any anomalies. Therefore, the decision on how long the simulation test should be conducted was done on the bases of the initial test runs. For example, it was decided not to record the data for a week duration because the model have some warmup time, and because of this the data collected might not reflect the true behaviour of the protocol. Therefore, it was decided to begin the test with a minimum of two weeks and increase the duration by two weeks to collect the data until consistency was achieved as shown in table 14.

The graphical representation of the performance results is depicted in Figure 6.11. The figure illustrates the duration of simulation run in weeks on X-axis, while the time (in minutes) it took to run the execution are illustrated on Y-axis. Underneath the simulation duration, a legend is presented with different colour codes for each number of participating federates. From the data shown in table 14 it was observed that there was a steady increase as the duration of the simulation run increases by weeks, with an exception of slight dip in the first two weeks run. This is because of the effects of warmup time. This increase can be noted from figure 6.11. This figure also represent similar behaviour for different number of federates. The reason for conducting the experiments with different number of federates will be discussed in next section of scalability testing. However, it is noted that having more federates in a distributed network results in similar performance. Finally, a linear forecast trend line is indicated by yellow dotted line for additional of two future periods to predict the execution time if the simulation ran longer than eight weeks.

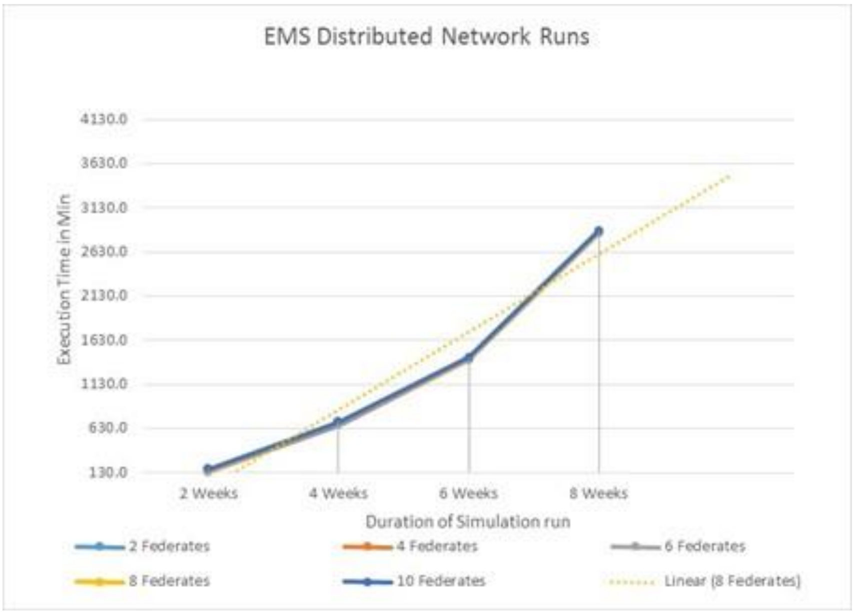


Figure 6.11: EMS Distributed Network Run

The distributed network model performance was also compared with single node execution on four-week run with a number of federates. The data collected is presented in table 15. This data is also graphically presented in figure 6.12. It was interesting to note that the single node or single pc execution for fewer federate was faster than running the simulation in a distributed environment. However, this was only true for up to certain number of federates. In the data and the graph presented, it can be observed that the graph takes a steep curve upwards after 6 federate and cross the execution time for networked simulation run. It should also be noted that there is no data available for ten-federate run for a single node execution. That is because the simulation for ten-federate was unable to successfully run on single node.

Federate	**2**	**4**	**6**	**8**	**10**
Single node	654.7	659.0	668.3	698.3	-
Distributed Network	665.7	675.2	685.4	694.3	703.0

Table 15: Single Vs Network Execution run

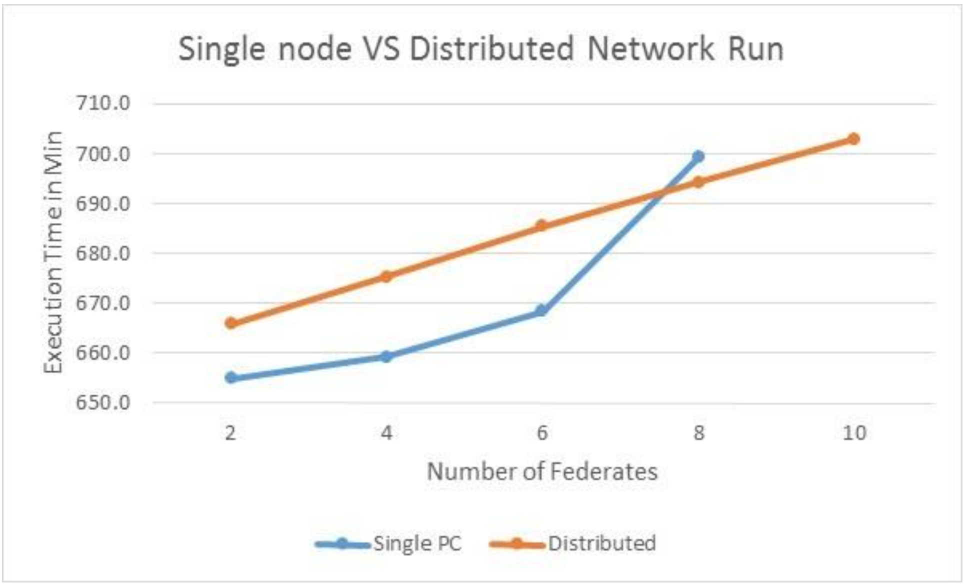


Figure 6.12: Single Vs Network Execution run

The reason for failure was further investigate and the cause of the problem was identified. The reason for the upward spike after the six federate and unsuccessful run of the ten federate was because of the shortage of main memory. Hence, a memory consumption table was created to identify the cause and break points.

Table 16 provides the figures of memory consumption and the CPU usage at the time on initialising federates. Note, this is not the memory consumption figures during the execution. Also, it is to be noted that this memory consumption includes the operating system and applications required to run this simulation i.e. approximately just over 50% of the memory is used by the operating system and other application such as repast simphony, while the remaining 50% is used at runtime. The CPU usage figures were hard to record but there were spikes of high usage at certain times i.e. at the start of the federate, therefore the data presented is the maximum average usage. Since, up until nine federate the CPU was underutilised therefore it was not considered as the root cause of problem.

No. Federates	Memory Consumption	CPU Usage
1	54.2%	60%
2	59.0%	64%
3	64.2%	68%
4	70.7%	66%
5	76.0%	70%
6	82.0%	73%
7	88.5%	72%
8	94.2%	73%
9	99.5%	75%

Table 16: Memory Consumption

However, up until six federate the execution of the simulation was smooth and it did outperform the networked execution run but thereafter the performance started to reduce dramatically. This is because up until six federate 82% of total memory was used at initialisation stage and the PC still had 18% of memory left to be used for the execution run. But when eight federates were executed 94.2% of the memory was used at the initialisation stage and a minor part of the auxiliary memory (i.e. page file) was used to run the simulation, which reduced the performance. In the case of nine or more federate, the federates require more than 100% primary memory for initialisation only, hence some of the federate were automatically moved to the auxiliary memory by the operating system. Therefore, the federation having more than eight federate, kept crashing because of the response time delay and were not possible to run. From all of the above it is derived that the single node execution can be limited because of memory and/or the complexity of the model, in this case it was the memory that was not sufficient enough to run the simulation. These findings also indicate that networked solution might have communication bandwidth as a bottle neck but it can overcome in case of large simulation with more federates involved. During these experiments the protocols implementation was tested for a longer time period and also on a larger simulation model. It was observed that these protocols did not contributed to any additional performance issues.

6.4.2 Scalability Testing

In section 6.3 the performance of the protocols was measured with the minimum number of federates. Further, tests were conducted with the larger implementation of London EMS case study, more federates were used as shown in table 14 to test if the protocols can be implemented with more than the minimum federates involved. Table 14 demonstrate the execution run for up to ten federates. Since the objective was to test the scalability of the protocol in scalable environment, therefore having ten federates was found sufficient to demonstrate the functioning of the proposed protocols. Figure 6.11 indicate that there is no or very little impact of scaling the federates and the execution time increased proportionally.

To examine the difference more closely each scaled up run is presented separately in figure 6.13. This figure contains four different experimental runs based on the duration of the runs. It was noticed that each of the simulation runs had a similar incline as the number of federate increased. Ideally this should be a flat and straight line. But, because as the number of federate increases the simulation complexity also increases i.e. each increase in federate represent an increase in the hospital model. Additionally, the area of ambulance services model also increases, and thus will encounter more emergencies i.e. the simulation becomes more computationally complex. This results in increased interoperability and more computation.

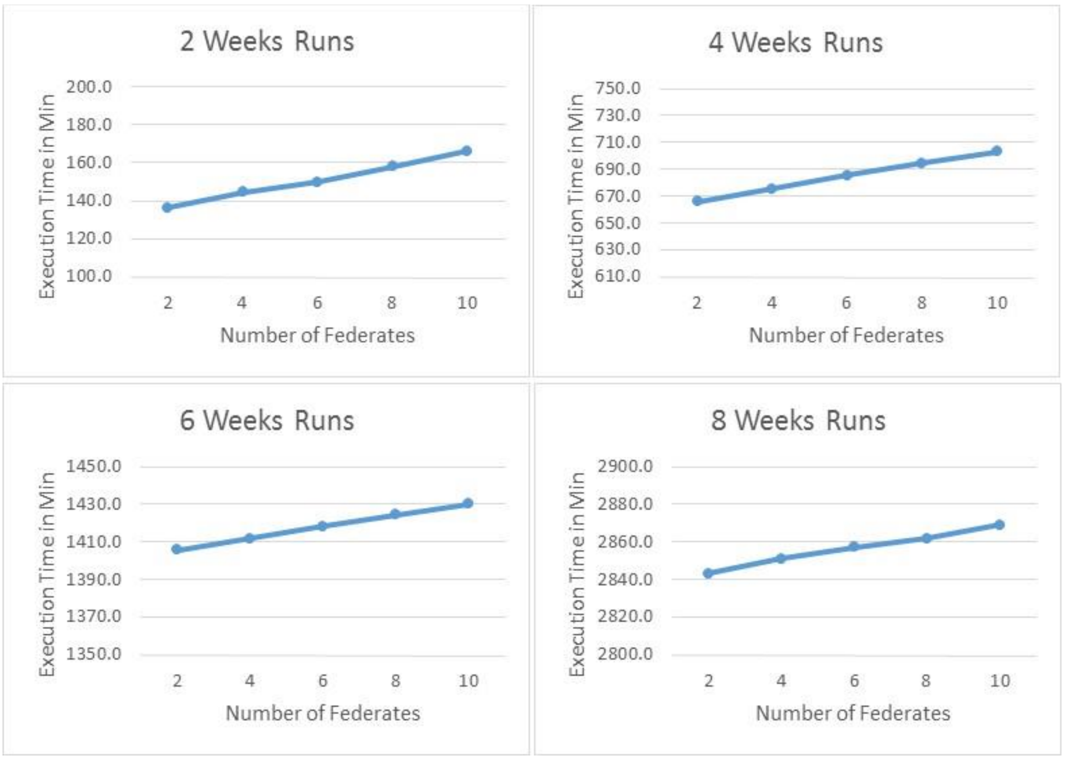


Figure 6.13: Scalability run results

In comparison with single node execution as illustrated in figure 6.12 and figure 6.11, the graph of network simulation is much more linear. The results of these experiment runs indicates that having a scalable model does not guarantee the same execution time. The simulation execution time is affected by the model complexity and the model response when the model is scaled.

6.5 Revisiting the DSI Framework

The number of plausible solutions to the interoperability problems were discussed in practical detail above. We have already acknowledged that distributed simulation cannot avoid interoperability issues and all such identified issues are listed in the IRMs. Therefore, a generic framework for addressing these identified interoperability problems (outlined in Chapter 4) is necessarily required. This is not only to address the interoperability problems but also to achieve better reusability and composability between distributed simulation models. Before this can be achieved, it is important to approve a standardised approach to addressing these interoperability issues as different solutions might create confusion for the practitioner and hinder in reusability. Therefore, this section discusses the best possible solution(s) to these interoperability problem for recommendation as part of the final DSI Framework based on both practical and theoretical analysis.

In revisiting the first interoperability problem, i.e., general entity transfer, discussion of the output of experiment runs in the previous section makes it clear there is little difference between the use of interaction classes and attribute classes, although attribute class implementation is slightly faster. But a theoretical analysis shows that entity passing is a one off event. There is no arithmetic operation to be performed once the entity is delivered. Any update on entity value or its attribute can occur at the receiving side by the model and as interactions are defined as non-persistent variables whereas attributes are persistent. Therefore, theoretically, using interaction classes for transferring entities is better suited as the entity object is not required later for any arithmetic or non- arithmetic update. Therefore, using interaction classes seems to be the better option for general entity transfer and is recommended for use.

The second interoperability problem, i.e., bounded receiving element, is related to the first problem. In fact, we can say that bounded receiving element problem is an extension of general entity transfer. Having established that using interaction classes for passing entities is the better approach, three different approaches were implemented each using interaction classes. All three listed approaches fulfil the requirements for bounded receiving element in Chapter 3. These three approaches were rigorously tested in different environments with different possible scenarios and with different RTIs but, as expected, the performance trend of each remained similar. Therefore, based on performance and implementation, the adaptive protocol can be recommended for use.

The third interoperability problem relates to multiple input prioritisation. This type of problem is very common in distributed simulation when more than two federates share or pass messages to each other at the same time. In a large distributed environment, there are greater possibilities for implementing solutions for concurrent message passing. Although experiment implementation was limited to entity passing, the protocols discussed can also be used for other scenarios. These other scenarios featured in implementation of the other interoperability problems involving concurrent access such as shared resources and shared data structure. The implementation discussed two different approaches, one specialised and the other dynamic. In practice, the performance of the specialised protocol is slightly better than the dynamic protocol. However, in theory dynamic protocol is more flexible, scalable, and reusable. Hence, the use of dynamic protocol is recommended to address this interoperability problem.

The fourth interoperability problem discussed related to shared resources between two or more federates. This interoperability problem is similar to shared data structure but, as defined above, there are differences and the same protocols and approach cannot be used for both problems. Here, four different approaches were implemented to evaluate performance and suitability. Because of RTI limitations the first protocol, i.e., continuous resource update, was struck out of the equation unless the limitation can be addressed. There is a huge performance delay and that is because of the RTI Callback function. This limitation is covered in more detail in the discussion of shared data structure below.

Based on performance figures, the next request protocol performed best in the Test 1 scenario, while in other cases it was inferior to the message queue protocol. The next request protocol also presents another drawback, i.e., resource time guarantee. A model cannot have a resource guarantee if multiple resource requests are generated from the other model, because a log of only one resource request per model is kept at any given time.

The message queue protocol is difficult to implement but provide resource scalability, time guarantee and no limitation to simulation techniques used. In fact, this protocol covers all the disadvantages of the protocols discussed above for resource sharing. This protocol is also suitable if the shared resource is maintained by a separate node. It provides a transparent layer between the model and the RIT.

The next resource event protocol is easy to implement in discrete event simulations, and also provides all the benefits of a message queue, but it is limited to simulation techniques using event list only because it uses the event list to schedule next release of the resource. This protocol also might not be suitable with other types of simulations. Therefore, in theory, the best recommended protocol for use to address general shared resource interoperability problems is the message queue protocol to maintain standardisation.

The fifth interoperability problem is related to shared event, which again is similar to general entity transfer. The only difference is the process of implementation, i.e., all federates must subscribe to the event, while possible implementations can be achieved using interaction or attribute classes. Using theoretical evaluation, events do not need further processing, so keeping them in a persistent state with RTI is of no use. Therefore, using interaction classes for this approach is recommended to address this interoperability issue.

Finally, the last interoperability problem is related to shared data structure. Three different protocols were discussed for this approach. Two of these have similar performance while one, i.e., full replication, has a somewhat lower performance due to RTI limitation. In the RTI implementation, callbacks are linked to time advance and there is no mechanism to generate a time guarantee callback without calling time advance. Hence, this protocol suffers from multiple time advance callbacks to RTI, which is responsible for a huge time delay, not only in synchronising the time of participating federates, but also in accessing and delivering all registered message events (even if none are used at that particular time interval). If RTI can present a separate callback method for particular registered variables, then this protocol could be ideal for use. Similar problems are encountered in continuous resource update protocol implementation, as discussed above. Unless this issue is resolved by RTI it is hard to recommend this approach. The remaining two protocols require a third central node to interact with the federate requesting shared data. In comparing the performance figures, central update is the solution most recommended for this interoperability problem.

6.6 Summary

This chapter has evaluated the different approaches discussed in the previous chapter. A standardised dedicated environment was put together to collect the experiment run results. A comprehensive performance review was conducted for each approach with different case studies having different parameters. Later all these approaches were critically evaluated both practically and theoretically to conclude the best practices for the DSI Framework.

The design, development, and testing of this framework brought forward some recommendations for improvement in the HLA standard and also recommended use of the proposed methodology for distributed simulation development. The next chapter summarises the research presented above and in the previous chapters.

Chapter 7: Conclusion

7.1 Overview

This concluding chapter of the thesis begins with a Research Summary, which sets out the defining motivation for this research: to assist HLA to fulfil its promise of solving interoperability problems in distributed simulation. The research summary explains that the research was stimulated by literature review, which revealed that while HLA assisted technical connectivity through structural data exchange measures it failed to provide the semantics of how data should be exchanged. This observation led to the objective of bridging the gap between integratability and interoperability by offering semantic solutions to the interoperability problems identified by IRMs.

The research summary below introduces development of the resulting DSI Framework, for integration with HLA, as the major contribution of this research. Validity of the hypothesis presented is confirmed as offering potentially significant advantages and encouragement to industry wishing to engage with distributed simulation methodology.

The research aim and objectives of the thesis are revisited by presenting five objectives and listing how each objective was met in the preceding chapters. Four contributions to science made by this thesis are described. Limitations to the work conducted in this thesis are set out. Finally, future research opportunities are presented to further simplify the modelling toolkit, expand case studies, and identify further problematic interoperability issues.

7.2 Thesis Overview and Findings

The motivation for this research began by seeking to understand why the simulation industry is not using the standard HLA and why the standard is not being promoted. The HLA objective was to promote reusability and address interoperability issues. But after interrogating several research studies and works from different authors, it was identified that HLA standard was not entirely fulfilling its promise to provide complete solutions to interoperability issues in distributed simulation. Close examination of the Wang et al. (2009) conceptual interoperability model identifies that HLA was able to provide interoperability solutions up to Level 2 (Figure 3.4), i.e., Syntactic Interoperability, while the BOM specification standards can provide interoperability from Levels 4 to 5, i.e., Pragmatic Interoperability and Dynamic Interoperability. The HLA implementation of Runtime Infrastructure (RTI) provides connectivity between federates at a technical level, and the HLA OMT specifications provide the structure of the data exchange but not the semantics of how data should be exchanged. Therefore, according to this paper's research, HLA was only achieving integratability at connection level and not real interoperability. This research identified the gap, i.e., at Level 3 Semantic Interoperability. Some argue that interoperability is a model responsibility, i.e., it should be addressed at Level 4 or the Pragmatic Level. Others are confused about the ownership of these interoperability issues. But this research believes that interoperability should be part of the HLA standard and composability should remain in the domain of model specific design.

The interoperability issues were identified in IRMs SISO-STD-006-2010. The objective of this interoperability standard was to identify and specify model interoperability capabilities and requirements. The scope of IRMs did not include semantic solutions for these issues and it was left to practitioners to identify candidate solutions for these problems. Therefore, further research was conducted on these interoperability issues to make them part of the HLA standard by introducing semantic solutions in OMT specifications.

To summarise the above, in addressing the hypothesis of testing the feasibility of a framework to tackle the IRM issues, it was observed that such implementation could also help bridge the gap between integratability and interoperability by offering semantic solutions to the interoperability problems. The major contribution of this research is the development of a generic framework to address these interoperability issues. This research believes that if HLA standard provides the semantic solutions to these interoperability problems, then it might also address issues like re-usability, composability, and increase in industrial use. Therefore, by combining HLA standards and the interoperability solutions, the distributed simulation industry might achieve a better interoperable distributed simulation standard.

To achieve this, an empirical research was conducted. In this study, the research stages followed were: propose the hypothesis; identify different approaches to address and evaluate the hypothesis; obtain results by applying the identified approaches iteratively; and finally, evaluate the hypothesis. The hypothesis was tested in a generic case study, which means it could be adopted by practitioners for use or for further research. This research had led to development of the generic DSI Framework and some design research for the development of distributed simulation projects.

To evaluate the proposed hypothesis, a detailed literature review was conducted to first investigate the gap in industry use that led to study of interoperability issues, followed by an explanation of the importance of interoperability, and finally the identification of an interoperability solution. To achieve this, six IRMs were identified. Although IRM defines four main interoperability issues, the first Type A has three sub-types. Therefore, this research addressed each separately and classified six interoperability issues. A total of seventeen different generic approaches were implemented covering all six interoperability issues, and some of these were also tested in a different network environment, with different RTIs and versions, and with different scenarios. The centre of implementation was the poRTIco RTI because it is open source, licence-free, and had the latest implementation of HLA Evolved when the research came to the design phases.

The research then indicated the need to list the “recommended” approaches to addressing these six interoperability issues to help the practitioner make more informed choices. Chapter 6 discussed all these approaches both practically and theoretically. Certain recommendations were made based on performance while others were made based on both performance and theoretical argument. The solutions to some interoperability issues were also tested in a specific case study published earlier (Nouman et al., 2013), in which the interoperability problems were tested in an EMS case study, using hybrid simulation technique.

The present research also identified different methodologies, including the DSEEP, but none of these reported a generic conceptual modelling to capture and address interoperability issues in distributed simulation. This research also proposes to fill that gap and provide modifications to existing methodologies to highlight and action the interoperability issues. The rationale behind the development of this methodology is grounded in the work presented by the DSEEP standard and conceptual modelling by Robinson (2014). A detailed literature review is presented in this research to identify, select, implement, evaluate, and verify the interoperability issues in a distributed simulation. In previous studies, interoperability issues were not given much importance and were not even included in the conceptualisation phase.

Another evaluation from this research was highlighted, which relates to RTI restrictions and flexibility. It was noted that very little technical documentation exists for implementation of RTIs and that this is common for both commercial and non-commercial based products. Available information was found to be scattered between IEEE standards and some from RTI publishers. This makes it harder for practitioners to understand and use the standards. During development of the DSI Framework, some approaches to design solutions for the interoperability issues were withdrawn because of RTI multiple message passing with a time guarantee within a time advance. Having a feature to avoid this will allow more options to be explored for proposing further approaches to these interoperability issues.

The evaluation of this research concludes that the proposed framework if used can bridge the gap and provide interoperability solutions to distributed simulation practitioners. Therefore, the hypothesis presented in this research is valid and demonstrates that the advantages of using the DSI Framework will benefit the distributed simulation industry.

7.3 Research aim and objectives revisited

The aim of this thesis is to investigate how to address interoperability issues in the distributed simulation environment to benefit simulation industry, and to provide recommendations (if any) for improvement in the HLA standard. To achieve the aim, five objectives were underlined.

- **Objective 1:** *Present the hypothesis and identify the interoperability issues as defined in the IRMs.*

  Based on initial research, Chapter 1 presented the research aim and hypothesis and also discussed the research methodology used for testing the hypothesis. While Chapter 2 introduced the interoperability issues defined by IRMs.

- **Objective 2**:*Develop an understanding of underlining standards, techniques, and industry approaches to identify if interoperability issues are hindering the use of distributed standards.*

  This objective was achieved in Chapter 2. To achieve this objective, a thorough literature review was conducted of different standards, industrial practices, industrial surveys, and work from different authors on distributed simulation, and these were discussed in depth. Chapter 2 also discussed different simulation techniques, simulation studies, and simulation models, and the different distributed simulation methodologies in practice. This chapter helped to establish the research gap and underline the requirements for resolving that gap, and information gained from the literature review provided the foundation for development of the generic DSI Framework.

- **Objective 3:** *Propose a framework to address interoperability issues.*

  Chapter 3 described the requirements for the DSI framework with the proposed methodology for modelling. The following Chapter 4 presented the proposed framework and the conceptual modelling of the different approaches used.

- **Objective 4**: *Experimentally test the DSI Framework*

  The solutions provided to all six interoperability issues were designed and developed using seventeen different approaches as detailed in Chapter 5. Chapter 6 presented the experimental runs and test results for evaluation.

- **Objective 5:** *Evaluate the performance of the DSI Framework and test the hypothesis*

  The results of the experimental runs were evaluated and recommendations for use were made in Chapter 6. Presentation of the research summary and acceptance of the hypothesis was completed in Chapter 7.

7.4 Framework Value

There are many challenges involved when creating a distributed simulation model. A single model can be created by using any CSP or open source simulation packages, but while creating a distributed simulation model for the same single model, it introduces interoperability challenges. Many different approaches were used to address this problem, but it is extremely difficult to capture the difference between the approaches and their implementation. Further, simulation practitioners and the vendors often find simulation interoperability techniques inaccessible. Therefore, to address this issue CSPI PDG was established to identify the common approaches to these problems. As part of this research IRM's were used to create a common frame of reference to access the capabilities of particular approaches and to help practitioners and vendors achieve solutions to complex interoperability problems. With a substantial effort by consulting, including CSP vendors and well-known practitioners, the distributed simulation interoperability problems were standardised. Therefore, a standard set of patterns was identified to encounter interoperability problems while developing distributed simulations.

The IRMs standard was intended to simplify the complexity and to assist practitioners. For both, solution developers and CSP vendors this standard was intended to identify solutions to their interoperability problem. This research presented a framework to provide possible solutions to these interoperability problems using a standard communication platform i.e. HLA. While developing the framework it was considered that the scope, purpose and target audience of this framework at least match with the IRM standards i.e. the framework should not be restricted to a particular CSP vendor or programming environment. However, the scope of this work is limited to the leading distributed simulation standard for communication i.e. HLA.

HLA standard offers many other features like reusability, portability, scalability etc. these are explained in more detail in chapter 2. After having all these benefits, still HLA is not widely used in the industry and practitioner restrain to use this standard because of its complexity. Therefore, CSPI PDG took a step further by identifying the interoperability issues in distributed simulations. This in itself was not sufficient to address the complexity of HLA. The IRM's were first presented in 2006 and after over a decade practitioner are still struggling. This research is a step forward to bridge gap between these interoperability problems and the HLA standards and provides an insight on how this could be achieved.

The focus behind this work was to keep it simple and easy for the practitioner. Therefore, the functioning of these protocols is almost isolated from the federate initialisation and the simulation model as described in figure 4.2. Most of the operation is happening within the IRM manager layer which is presented in this research. This research also enhances the practitioner understanding by simplifying the simulation into layers. This framework also guides the practitioner on selection of different approaches for developing a distributed simulation. This framework is not restricted to be only used with standard practices used in the industry. Infect, this could be used in conjunction with either. But using the standard approach will widen the prospects for reusability and portability. It is clear that reusing a model can save cost both time and money. But this is not always possible, because to reuse a model, minor modifications are required, but if a DS model is developed using a nonstandard approach, then it not only becomes difficult for successor practitioners to understand the model but also it might not be compatible with the other environment. Having a standard approach will help to overcome these problems.

During this research a number of published related work was identified in chapter 2. Different authors have provided solutions to these problems, but if studied in detail, they are limited to a specific problem. While, the proposed solutions can also be adopted for those problems. The solutions provided in the research are based on the most recent technologies used in the industry and the proposed solutions is derived on these technologies. Therefore, this framework also inherits the benefits of underlying technologies. The use of the proposed framework will also promote the use of standards in the industry. This will not only further help to improve the standard, but also widen the scope of implementation.

The industry is expecting advancement from DS especially for large-scale simulation projects. Having these standards in place, simulation industry could benefit greatly with the new emerging technologies such as cloud computing. The benefits could be unlimited, on-demand access to multiple computing resources (Chaudhry et al, 2015). To overcome the significant technical challenges faced by cloud computing, it is important that standard practices are adopted to increase the scope of reusability and address the interoperability issues. The work presented in Chaudhry et al (2015) also tested one of the proposed protocols over the cloud.

Although, the proposed protocols and the case study used in this research uses the open source software's and packages, but the work is not limited to the given environment. Some of the work had been tested to be working in different commercial RTI environments such as the most popular Pitch and MAK RTI, similarly two other commercial simulation packages i.e. Simul8 and Anylogic were also used for testing purposes. This was not discussed earlier in the thesis, but it was tested during this research. Both of these simulation packages were tested with a general entity transfer Type A.1 IRM.

Similarly, development environment is not restricted as long as the detail functioning and the sequence of events are known. Therefore, in this thesis the focus was on explaining the details of individual protocol detached from any specific programming language. It is understood that all the CSP's do not use a particular language, therefore to scope this framework a practitioner should understand, how they can use proposed protocol and then translate it into the required programming language. A simple test was conducted at the beginning of the research with Simul8 using C++ language. The practitioner must consult to the CSP manual to identify the available options. Some CSPs provide this as additional facility and do not ship it with the standard version.

PoRTIco, on the other hand is not limited for use in open source simulation packages. As described earlier, this was also tested with some CSPs. There are commercial applications using this RTI e.g. Titan. IM the next wave in a virtual simulation, Unity3D a powerful cross-platform 3D engine and a user-friendly development environment, alongside application packages like AutoCAD and Simulink for Matlab can also benefit from this RTI. However, there are still some compatibility issues between different RTI's which could be used for further research to investigate.

Distributing a simulation require a simulation package, RTI and the interoperability solution. Once a simulation package is identified, i.e. a package that can support external script/API's or to have capability to use code within the simulation. If coding or the programming languages are not supported, then a bridging API could be designed. The selection of a simulation package can also be influenced by the proposed RTI, because some RTI's might provide limited support. Finally, the selection of interoperability solution. This thesis provides the detail functioning of each protocol in the simplest way possible i.e. using a federate with a simple simulation model. It is also not necessary, that both simulation package and RTI must use the same programming language. In case of Simul8 example, this was tested by creating an interface to communicate between the RTI implementation (using C++) and Simul8 (Visual Logic).

Finally, the required interoperability problems must be identified before selecting any interoperability solution. A thorough analysis must be conducted to understand the requirements of each interoperability problem and then match these requirements with the proposed protocol. Selection of Protocols (section 5.4.4) presents an example exercise on how these details can be compared to selecting the necessary interoperability solution. In this example, three different IRM's were used and compared with the proposed protocols to identify the best possible solution for the problem. For the practinior support and guidance all the protocol source code used in the thesis is also made available for the practicitioners on Github at the following link (https://github.com/atharnoumangit/IRM-Protocol). Although, this might require setting up parameters but default settings are built in the sample code.

7.5 Research Contribution

After establishing the interoperability issues faced by the practitioners while distributing the simulation models, different protocols were proposed to address these interoperability challenges. After conducting experiment runs on these proposed protocols they were evaluated and a propose solution was concluded to address the interoperability issues presented by IRMs. This research also uncovered some other contribution which are listed below:

1. The major contribution of this research is the DSI Framework, to enable simulation modellers and vendors to satisfactorily address the interoperability requirements of their distributed simulation models. This framework provides a set of protocols that can be used by the practitioners to address the interoperability between their distributed simulation models. This research also helps the beginners (in the field of *distributed* simulation) to understand the challenges faced while distributing a simulation.

   The core objective of the HLA standard was to achieve interoperability, but this research and the IRMs have clearly identified that HLA standards have not completely met these objectives specially issues related at semantic level. Therefore, this research proposes that these semantic recommendations become part of the HLA OMT standardised specification to help practitioners.

Only if the HLA OMT can specify the structure of interoperability and data exchange, then it can reach the semantic level. The DSI Framework provides a generic solution to each interoperability issue, enabling practitioners to use the recommendations to address known interoperability problems. This research also highlighted the relationship between interoperability, reusability, and composability. The proposed framework can help the HLA standard fulfil its promise of achieving interoperability, and standard interoperability semantics will promote reusability in distributed simulation, encouraging potential increased support from industry.

2. The second contribution of this research is to facilitate capturing of interoperability requirements at a semantic level. Identifying the requirements during the conceptual modelling phase will allow to prepare validation and verification test condition. The proposed distributed simulation modelling methodology highlights the importance of interoperability, which was a subject missing from all extant simulation methodologies. One reason for this absence is because a majority of methodologies do not address distributed simulation but only discuss modelling of single simulations model executed on a single node, i.e., non-distributed. A major justification for distributing a simulation is because of system complexity and resource limitations, but practitioners were not being provided with the proper tools and methodologies for modelling distributed simulation. This research will therefore help practitioners to address these challenges, during design and development of a distributed simulation.

3. The third contribution of this research is the identification of the requirement for new callback mechanisms in HLA Standard. This emanated from framework development and the experiment runs. It was noted during the experiment runs that the majority of time was consumed by model time synchronisation and time advancement because RTI provides a guarantee to deliver all messages before time advance. This contribution highlights the need to introduce additional methods for callback in RTI implementation that could only guarantee synchronised delivery of messages without time advance. This could be achieved in an open source RTI, but changing this in a particular RTI will defeat the objective of the standards. Therefore, this research proposes additional callback methods in HLA federate interface specifications, which can also open another dimension of research into better performing solutions. The other proposed change related to this change is to have an additional method whereby synchronised time advance can be forced without checking and delivering all messages.

The proposed framework can be introduced as an extension to SISO Standard SISO-STD-006-2010. As mentioned above this standard present a set of templates or patterns for specific interoperability problems faced by the industry experts in distributed simulation models. The work presented in this thesis, propose a solution to these interoperability problems therefore it will be best to present an extension to this standard as SISO-STD-006-2010-2. The proposed framework uses only HLA standard therefore this extension will also produce a similar HLA OMT specification for this framework within this standard extension as presented in HLA Object Model Template Specification (IEEE1516.2.2010).

4. The final contribution made by this research concerned making models more integratable and reusable. Models reusability and integratability can seriously be affected when each model uses custom build method to address interoperability. The solution to this is achieved by bridging the gap between integratability and interoperability by offering semantic solutions to the interoperability problems. Using a standards-based approach to address interoperability issues, through a standard communication platform such as HLA, enables model design to become more integrated and reusable.

7.6 Research Limitations

This research contributed to addressing the interoperability issues identified by IRMs. The research was limited to the interoperability issues identified by SISO, and there could be other interoperability issues specific to a certain problem domain. The author does not claim to have exhausted all possible interoperability problems other than IRMs because it is extremely difficult to capture all scenarios.

The framework uses generic case studies, and only two of them were tested in a specific case study, but number of experimental runs were conducted to depict different system behaviour. Yet again, it is not possible to capture all the scenarios due to time limitation. Similarly, these proposed recommendations were mainly tested using one open source RTI and in few cases comparison with one commercial RTI. Due to time constraints all the RTI can't be tested and specially the commercial RTI vendors do release frequent versions. It can become extremely difficult to keep track of the performance comparison.

Chapter 6 discussed the individual limitations of each approach, but in terms of testing almost all the experimental runs were made on a dedicated network because it would be hard to calculate alternative performance delay. As discussed in Chapter 6, performance will vary on different networks, but the performance graphs presented in this research were based on experiment runs over a dedicated network. Also, due to the huge number of experiment runs, it was not possible to collect data for more than five weeks of the run.

7.7 Future Work

This research has shown how investigation into the design and development of distributed simulation can produce work that is of benefit to practitioners. The target audience, i.e., the end-user, is considered to comprise industry experts. These industry experts are expected to have a knowledge of M&S, and underlying technologies (e.g., Java or C++), and an understanding of HLA standard, that is, RTI. In earlier discussion it was highlighted that RTI documentation is not as good as it should be. Therefore, to promote the use of HLA standards, it would be ideal to develop higher-level tools, or a toolkit, that can hide the complexity of distributing the simulation and enable the modeller to concentrate only on the model itself.

Because of time limitation, all the researched approaches were tested only on generic case studies. It would be rewarding to test different proposed approaches on specific case studies to compare and re-evaluate the performance. Since the case studies were very generic, they had no or very little effect on the overall performance.

Furthermore, this research was limited to the interoperability issues identified by the IRM. Further research could be conducted to identify more interoperability issues, which could be more specific to a problem, for appending to the recommendations.

7.8 Summary

The Research Summary that begins this chapter commenced by stating the motivation for this research was to enable HLA to better fulfil its objective of overcoming interoperability problems in distributed simulation.

Literature review had exposed that HLA was not providing complete solutions to interoperability issues and while IRMs identified interoperability problems they did not offer semantic solutions to resolve them.

The hypothesis put to the test, therefore, was the feasible of defining a framework that implements the issues identified by SISO standard for COTS Simulation Package IRMs (SISO-STD-006-2010), which could be effectively adopted by the HLA standard. The stages of this research study were listed from hypothesis proposal to experiment evaluation, involving a generic case study and seventeen different approaches to implementing six IRMs.

It was noted that some potential implementations were withdrawn due to RTI limitations, giving rise to recommendations for changes to RTI features. This chapter confirmed through evaluation of the experimental tests that the validity of the hypothesis was confirmed and continued by listing the five objectives underpinning the research aim, and the chapters in which each were addressed.

The contributions made by his research were stated as threefold: capturing interoperability requirements at semantic level for integration with HLA to enable modellers and vendors to address these issues; the proposed distributed simulation modelling methodology, which establishes interoperability as a core feature of simulation design and development; identifying the need for additional callback methods in RTI implementation to enable improved interoperability solutions; and finally making models more integratable and reusable.

Limitations of this research were noted as not covering unknown potential specific interoperation problems; the majority of case studies being generic rather than specific; limits on the number of RTI products tested; and tests being conducted on a dedicated network limited to five weeks' duration.

Future research suggestions included development of a higher level toolkit to hide complexity for the modeller; further testing of the proposed approaches using specific case studies, and identification of further case-specific interoperability problems.

References

Al-Zoubi, K. and Gabriel, W. (2011) "Distributed Simulation Using Restful Interoperability Simulation Environment (RISE) Middleware". *Intelligent-Based Systems engineering,* Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, pp. 129-157, ISBN: 978-3-642-17931-0.

Anagnostou, A. (2014). “A Distributed Simulation Methodology for Large-scale Hybrid Modelling and Simulation of Emergency Medical Services.”, PhD. Thesis, Department of Information Systems and Computing, Brunel University London. [online] Bura.brunel.ac.uk. Available at: http://bura.brunel.ac.uk/handle/2438/11218 (Accessed 14 Mar. 2017).

Anagnostou, A. Taylor, S.J.E. (2017), “A Distributed Simulation Methodological Framework for OR/MS Applications”, Simulation Modelling Practice and Theory, Vol. 70, pp. 101-119.

Anon, (2011), “M&S VV&A RPG Core Document: VV&A of Federations”, M&SCO Modeling & Simulation Coordination Office, [online] Available at: https://vva.msco.mil/default.htm?role/Federation/default.htm (Accessed 18 Feb. 2017).

Arbez, G., and Birta, L.G. (2010) “The ABCmod Conceptual Modeling Framework. In Conceptual Modeling for Discrete-Event Simulation”, Chapman and Hall /CRC, Boca Raton, FL, pp. 133-178.

Balci, O. and Robert G.S., (1984) “A Bibliography on the Credibility Assessment and Validation of Simulation and Mathematical Models”. Simuletter, Vol. 15, No. 3, pp. 15-27.

Balci, O. (1994). “Validation, Verification, and Testing Techniques Throughout the Life Cycle of a Simulation Study.” *Annals of Operations Research,* Vol. 53, pp. 121-173.

Balci, O., Arthur, J.D. and Nance, R.E. (2008) “Accomplishing Reuse with a Simulation Conceptual Model.” *In Proceedings of the 40th Winter Simulation Conference,* Edited by S.J. Mason, R.R. Hill, L. Monch, O. Rose, T. Jefferson and J.W. Fowler, pp. 959- 965, Miami, FL, December 07-10.

Balci, O., Arthur, J.D. and Ormsby, W.F. (2011) “Achieving reusability and composability with a simulation conceptual model.” *Journal of Simulation,* Vol 5, No.3, pp. 157–165.

Balci, O. (2016) “Introduction to modeling and simulation”. ACM SIGSIM Modeling and Simulation Knowledge Repository (MSKRR) Courseware. [online] Available at: https://www.acm-sigsim-mskr.org/Courseware/Balci/introToMS.htm. (Accessed 20 Apr. 2016)

Banks, J., Carson, J.S., Nelson, B.L. and Nicol, D.M. (2009). “Discrete-Event System Simulation”, 5th edition. Printice Hall, Upper Saddle River, NJ, ISBN: 978-0-136-06212-7.

Barros, F., Wang, M.H., Prähofer, H. and Hu, X., “Proceedings of the Symposium on Theory of Modeling & Simulation”, in proceedings of DEVS Integrative M&S Symposium (DEVS '15), Society for Computer Simulation International, San Diego, CA, USA, 9-16.

Boer, C.A. and Verbraeck, A. (2003) “Distributed Simulation with COTS Simulation Packages”. *In Proceedings of the 2003 Winter Simulation Conference,* Edited by S.Chick, P. J. Sanchez, D. Ferrin and D. J. Morrice, 829-837. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Boer, C.A. (2005) "Distributed Simulation In Industry". Submitted as a Ph.D. Thesis in Erasmus University Rotterdam, Print, ISBN: 978-905892-093-5.

Boer, C. A., Bruin, A. D. and Verbraeck, A. (2006) "Distributed Simulation in Industry - A Survey Part 2 - Experts on Distributed Simulation," *Proceedings of the 2006 Winter Simulation Conference,* Monterey, CA, pp. 1061-1068.

Boer, C. A., Bruin, A. D. and Verbraeck A. (2008) "Distributed simulation in industry - a survey Part 3 - the HLA standard in industry," *2008 Winter Simulation Conference*, Austin, TX, 2008, pp. 1094-1102.

Borah, J.J. (2006) "SISO-REF-017-2006 Simulation Conceptual Modeling (SCM) SG final report". [online] Available at: http://www.sisostds.org (Accessed 14 Mar. 2016).

Borshchev, A. and Filippov,A (2004) "From System Dynamics and Discrete Event to Practical Agent Based Modeling: Reasons, Techniques, Tools", *The 22nd International Conference of the System Dynamics Society*, July 25 - 29, Oxford, England.

Brooks, R.J. and Robinson, S. (2001). "Simulation and Inventory Control". Basingstoke: Palgrave Macmillan, Reproduced with Permission of Palgrave Macmillan, ISBN: 978-0-333-79429-6.

Bryant, R.E. (1977) "Simulation of Packet Communications Architecture Computer Systems." MIT-LCS-TR-188, *Massachusetts Institute of Technology,* Cambridge, MA, USA.

Bryman, A. and Bell, E (2007) "Business Research Methods." 2nd edition. New York: Oxford University Press Inc.

Buxton, J.N. and Laski, J.G. (1962) "Control and simulation language". *In Computer Iournal,* Vol 5, pp. 194-199.

Cardoso, L., Marins, F., Quintas, C., Portela, F., Santos, M., Abelha, A. and Machado, J. (2014) "Interoperability in Healthcare", In Cloud Computing Applications for Quality Health Care Delivery, pp. 78-101.

Calvin, J.M., Dickens, A., Gaines, B., Metzger, P., Miller, D. and Owen, D. (1993) "The SIMNET Virtual world architecture". *In Proceedings of IEEE Virtual Reality Annual International Symposium,* Seattle, WA, 1993, pp. 450-455.

Chandy, K.M., and Misra, J. (1979) "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs." *In IEEE Transactions on Software Engineering,* Vol. SE-5, No. 5, pp. 440-452, Sept. 1979.

Chaudhry, NR., Nouman, A., Anagnostou, A., Taylor, SJE., (2015), "WS-PGRADE workflows for cloud-based distributed simulation", 2015 Winter Simulation Conference (WSC), Huntington Beach, CA, 2015, pp. 3180-3181.

Collier, N., Howe, T. and North, M.J. (2003) "Onward and upward: The transition to Repast 2.0.", *In Proceedings of the first annual North American Association for Computational Social and Organizational Science conference*. Edited by: Carley K. Carnegie Mellon University, Pittsburgh, Electronic Proceedings

Creswell, J. W. (2003) "Research Design. Qualitative, Quantitative, and Mixed Methods Approaches.", Second edition, Thousand Oaks, California, USA: SAGE Publication.

Dahl, O. and Nygaard, K. (1966) ''SIMULA: an ALGOL-Based Simulation Language''. *In Communications of the ACM*, Vol. 9, No.9, pp.671–678.

Dahmann, J.S., Kuhl, F., and Weatherly, R. (1998) "Standards for Simulation: As Simple as Possible But Not Simpler: The High Level Architecture for Simulation." Simulation, Journal, Vol. 71, No. 6. pp. 378 - 387

Davis, P. K. and Robert H. A. (2003) "Improving the composability of Department of Defense models and simulations." RAND National Defense Research Institute. [online] Available at: https://www.rand.org/content/dam/rand/pubs/monographs/2004/RAND_MG101.pdf (Accessed 20 April. 2015)

Defense Modeling and Simulation Office (DMSO). (1995) “DoD Modeling and Simulation Master Plan”, U.S. Department of Defense, October 1995, (Accessed 26 Jan. 2015).

DMSO. (1998a) "High Level Architecture Object Rules v1.3", Technical Report, Defense Modeling and Simulation Office, Washington DC, USA, [online] Available at: http://www.hla.dmso.mil/ (Accessed 8 Sep. 2013).

DMSO. (1998b) "High Level Architecture Interface Specification v1.3", Technical Report, Defense Modeling and Simulation Office, Washington DC, USA, [online] Available at: http://www.hla.dmso.mil/_(Accessed 8 Sep. 2013).

DMSO. (1998c) "High Level Architecture Object Model Template v1.3", Technical Report, Defense Modeling and Simulation Office, Washington DC, USA, [online] Available at: http://www.hla.dmso.mil/.(Accessed 8 Sep. 2013).

Endres, A., and Rombach, D. (2003) “A Handbook of Software and System Engineering: Empirical Observations, Laws and Theories: (The Fraunhofer IESE series on software engineering)” 1st Edition, Addison Wesley, Harlow, England, ISBN: 978-0-321-15420-0.

Engel,A. (2010) “Verification, validation, and testing of engineered systems”, 1st Edition, Andrew P. Sage, Ed.: A John Wiley & Sons, Inc., Publication.

Fiddy, E., Bright, J.G. and Hurrion, R.D. (1981) ‘‘SEE-WHY: interactive simulation on the screen’’. *In Proceedings of the Institute of Mechanical Engineers* C293/81, pp. 167–172.

Fmi-standard.org, (2017). Functional Mock-up Interface. [online] Available at: http://fmi-standard.org/ [Accessed 18 Jan. 2017].

Forrester, J. (1958) “Industrial Dynamics: A Major Breakthrough for Decision Makers.” Harvard Business Review, Vol. 36, No. 4, pp. 37-66.

Fujimoto R. M. (1987), “Performance Measurements of Distributed Simulation Strategies,” Transactions of the Society for Computer Simulation 6(2): 89-132.

Fujimoto, R. (2000) “Parallel and distributed simulation systems”. New York, Wiley, ISBN: 0-471-18383-0.

Fujimoto, R., Bock, C., Chen, W., Page, E. and Panchal, J. (2017) “Research Challenges in Modeling and Simulation for Engineering Complex Systems”. Cham, Springer, ISBN 978-3-319-58544-4.

• Feldkamp, N., Bergmann, S., and Strassburger, S., "Visual analytics of manufacturing simulation data," 2015 Winter Simulation Conference (WSC), Huntington Beach, CA, 2015, pp. 779-790.

Garro, A. and Falcone, A., (2015) “On the integration of HLA and FMI for supporting interoperability and reusability in distributed simulation”. *In Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium (DEVS '15)*, Fernando Barros, Moon Ho Wang, Herbert Prähofer, and Xiaolin Hu (Eds.). Society for Computer Simulation International, San Diego, CA, USA, pp. 9-16.

Garro, A., Falcone, A., Chaudhry, N.R., Salah, O.-A., Anagnostou, A. and Taylor, S.J.E. (2015) “A Prototype HLA Development Kit: Results from the 2015 Simulation Exploration Experience.” *In Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM PADS '15).* ACM, New York, NY, USA, pp. 45-46.

Gogg, T. and Mott, J. (1992) “Improve Quality and Productivity with simulation.” JMI Consulting Group, Palos Verdes Pnsl., CA, ISBN-13: 978-1882229031.

Gordon, G., (1961) “A General Purpose Systems Simulation Program.” *In Proceedings of EJCC,* McMillan NY, Washington D.C., pp. 87-104.

Guizzardi, G. and Wagner, G., (2012) “Tutorial: Conceptual Simulation Modeling with Onto-UML”. *In Proceedings of the 2012 Winter Simulation Conference,* Edited by C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A.M. Uhrmacher. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc, pp. 1-15.

Gutiérrez, M. and Leone, H., (2013) “Composability Model in a Distributed Simulation Environment for Supply Chain”. Iberoamerican Journal of Industrial Engineering. Vol 5, pp.55-69.

Hakiri, A., Berthou, P. and Gayroud,T. (2010) “Addressing the Challenge of Distributed Interactive Simulation With Data Distribution Service.”, *In Proceedings of the 2010 Euro Simulation Interoperability Workshop*.

Harkrider,S. and Lunceford, H.W. (1999) "Modeling and Simulation Composability," *In Proceedings of the Interservice/Industry Training, Simulation and Education Conference*, Orlando, FL.

Hills P.R., (1973) “An introduction to simulation Using SIMULA”, NCC Publication No. 5-s. Norwegian Computing center, Oslo.

Hofmann MA. (2004) "Challenges of model interoperation in military simulations". *In Journal of Simulation*, Vol. 80, No.12, pp. 659-667.

Hollenbach, J. W. (2009) “Inconsistency, Neglect, and Confusion; A Historical Review of DoD Distributed Simulation Architecture Policies”. *In Proceedings of the Spring Simulation Interoperability Workshop*, Spring, San Diego, CA. www.sisostds.org

Hoover, S.V. and Perry, R.F. (1990) “Simulation: A Problem-Solving Approach”. Addison-Wesley, Reading, MA. ISBN: 978-0-201-16880-8.

Hughes, C. (2016) “Qualitative and Quantitative approaches.” The University of Earwick,. N.p.,[online] Available at: http://www2.warwick.ac.uk/fac/soc/sociology/staff/hughes/researchprocess/quantitative_and_qualitative_approaches.docx, (Accessed on Web. 15 Feb. 2016).

IEEE. (2000a) "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules (IEEE Std 1516-2000)", Technical Report, Institute of Electrical and Electronics Engineers, Inc., (Accessed on Web. 15 Mar. 2014).

IEEE. (2000b), "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification (IEEE Std 1516.1-2000) ", Technical Report. Institute of Electrical and Electronics Engineers, Inc., (Accessed on Web. 15 Mar. 2014).

IEEE. (2000c), "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification (IEEE Std 1516.2-2000) ", Technical Report, Institute of Electrical and Electronics Engineers, Inc., (Accessed on Web. 15 Mar. 2014).

IEEE. (2010), "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification", IEEE Standard 1516.1, 2010. (Accessed on Web. 10 Dec. 2014).

IEEE (2013), "IEEE Standard for Recommended Practice for Distributed Simulation Engineering and Execution Process Multi-Architecture Overlay (DMAO), IEEE Standard 1730, 2013, (Accessed on Web. 10 Dec. 2014).

Ingalls, R. G. 2013. "Introduction to simulation". *In Proceedings of the 2013 Winter Simulation Conference: Simulation: Making Decisions in a Complex World* (WSC '13). IEEE Press, Piscataway, NJ, USA, 291-305.

Jagdev, H. S. and Thoben K.-D. (2001) "Anatomy of enterprise collaborations." Journal of Production Planning & Control, Vol. 12, No.5, pp. 437-451.

Johnson, C. (2003) "What is research in computing science? Teaching notes", Department of Computer Science, University of Glasgow. Available online *http://www.dcs.gla.ac.uk/~johnson/teaching/research_skills/research.html*. (Accessed on Web. 28th March 2014).

Jones, S.G., Ashby, A.J., Momin, S.R. and Naidoo, A. (2010) “Spatial Implications Associated with Using Euclidean Distance Measurements and Geographic Centroid Imputation in Health Care Research.” Health Services Research, Vol. 45, No.1, pp. 316-327.

Karagoz, N.A. and Demirors, O. (2011) “Conceptual Modeling Notations and Techniques: In Conceptual Modeling for Discrete-Event Simulation”. Chapman and Hall /CRC, Boca Raton, FL, pp. 179-209.

Kasputis, S. (2000) "Composable Simulations,” *In Proceedings of the 2010 Winter Simulation Conference*, Orlando, USA, pp. 1577–1584.

Kneebone, R., Arora, S., King, D., Bello, F., Sevdalis, N., Kassab, E., Aggarwal, R., Darzi, A., Nestel, D. (2010) “Distributed simulation - Accessible immersive training”, Medical teacher, Vol: 32, pp. 65-70.

Knight, P., Corder, A., Liedel, R., Giddens, J., Drake, R., Jenkins, C. and Agarwal, P. (2002) “Evaluation of run time infrastructure (RTI) implementations”. Paper presented at *Huntsville Simulation Conference*, Huntswille, AL.

Kostelic, C. (2017). “Applying the Levels of Conceptual Interoperability Model to a Digital Library Ecosystem – a Case Study”. *In Proceeding of International Conference on Dublin Core and Metadata Applications*, The Washington, D.C., USA.

Kuhl, F., Weatherly, R., and Dahmann, J. (1999) “Creating Computer Simulation Systems: An Introduction to the High Level Architecture”, New Jersey, USA: Prentice Hall, ISBN: 978-0-130-2-2511-5.

Law, A.M. and Kelton, W.D. (1999) “Simulation Modeling and Analysis”, 3rd edition. New York: McGraw- Hill. ISBN-13: 978-0070582903.

Law, A.M. (2006) “Simulation Modeling and Analysis”, 4th edition. McGraw-Hill, New York. ISBN-13: 978-0070667334.

Law, A.M. (2014) “Simulation Modeling and Analysis”. McGraw-Hill Education; 5th edition, ISBN: 978-0073401324.

Leal, G., Guédria, W. and Panetto, H. (2017) “Assessing Interoperability Requirements in Networked Enterprises: A Model-Based System Engineering Approach”. INSIGHT, Vol 20, No.4, pp.15-18.

Low. M.Y.H., Gan. B.P., and Wei.J. (2006) “Shared State Synchronization for HLA-Based Distributed Simulation.” *In Journal of Simulation*, Vol 82, No. 8, pp. 511 – 521.

Loper, M. (2015) “Modeling & simulation in the systems engineering life cycle: Core concepts and accompanying lectures”, Series: Simulation Foundations, Methods and Applications. Springer, ISBN 978-1-4471-5634-5.

Loureiro, E., Nixon, P. and Dobson, S. (2010) "Adaptive Management of Shared Resource Pools with Decentralized Optimization and Epidemics," 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, Pisa, pp. 51-58.

Mahmood, I. (2013) “A verification framework for component based modeling and simulation: putting the pieces together”. Ph.D. Thesis, KTH School of Information and Communication Technology.

Mehl, H., and Hammes, S. (1993) “Shared variables in distributed simulation.” *In Proceedings of the 7thWorkshop on Parallel and Distributed Simulation*, pp. 68-75.

Mehl, H. and Hammes, S. (1995) “How to integrate shared variables in distributed simulation?” ACM SIGSIM Simulation Digest, Vol. 25, No.2, pp.14-41.

Michael, C. Fu. (2002) “Optimization for Simulation: Theory vs. Practice.” *INFORMS Journal on Computing*, Vol. 14, No.3, pp. 192-215.

Mustafee, N. and Taylor, S.J.E. (2006) "Investigating Distributed Simulation with COTS Simulation Packages: Experiences with Simul8 and the HLA." In *Proceedings of the Operational Research Society 3rd Simulation Workshop (SW06),* pp. 33-42.

Mustafee, N., Taylor, S.J.E., Katsialaki, K. and Brailsford, S. (2009) "Facilitating the Analysis of a UK National Blood Service Supply Chain Using Distributed Simulation." *Simulation,* Vol. 85, No. 2, pp.113-128.

Nance, R.E. (1994) "The Conical Methodology and the Evolution of Simulation Model Development". Annals of Operations Research Vol 53, pp. 1-45.

Nance, R.E. (1995) "Simulation Programming Languages: An Abridged History." *In Proceedings of the 27th Winter Simulation Conference,* Edited by C. Alexopoulos, K. Kang, W.R. Lilegdon and D. Goldsman, pp. 1307-1313, Arlington, VA, December 3-6.

North, M.J., Howe, T.R., Collier, N.T. and Vos R.J. (2005) "The Repast Simphony runtime system". *In Proceedings of the agent 2005 conference on generative social processes, models, and mechanisms.* Edited by: Macal C, North M, Sallach D. Argonne National Laboratory, Argonne, IL, pp.151–158.

North, M.J. and Macal, C.M. (2007) "Managing Business Complexity." New York: Oxford University Press, Inc, ISBN: 9780195172119.

North, M.J., Collier, N.T., Ozik, J., Tatara, E., Altaweel, M., Macal, C.M., Bragen, M. and Sydelko, P. (2013) "Complex Adaptive Systems Modeling with Repast Simphony," In Complex Adaptive Systems Modeling, Springer, Heidelberg, FRG, ISSN: 2194-3206.

Nouman, A., Anagnostou, A. and Taylor, S. J. E. (2013) "Developing a Distributed Agent-Based and DES Simulation Using poRTIco and Repast," *IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications,* Delft, 2013, pp. 97-104.

Oberkampf, W.L. and Roy C.J. (2010) *“Verification and Validation In Scientific Computing”*, 1st edition, Cambridge University Press, ISBN: 978-1-139-49176-1.

Onggo, B.S.S. (2007) “Towards a unified conceptual model representation”. Working Paper. The Department of Management Science, Lancaster University, revised in 2017, - Lancaster EPrints. [online] Available at: http://eprints.lancs.ac.uk/48913/ [Accessed 02 Jan. 2017].

Oses, N., Pidd, M., and Brooks, R. J. (2003) "Critical Issues in the Development of Component- Based Discrete Simulation", Technical Report, LUMS Working Paper, The Department of Management Science, Lancaster University, UK.

Overstreet, C.M. and Nance, R. E. (1985) “A specification language to assist in analysis of discrete event simulation models”. Commun. ACM Vol. 28, No. 2 , pp.190–201.

Paul, R.J. and Taylor, S.J.E. (2002) “What Use is Model Reuse: Is There a Crook at the End of the Rainbow?” In *Proceedings of the 34th Winter Simulation Conference*, Edited by E. Yucesan, C.H. Chen, J.L. Snowdon and J.M. Charnes, pp. 648-652, San Diego, CA, December, pp.08-11.

Page, E. H. and Opper, J. M. (1999) "Observations on the complexity of composable simulation," *in Proceedings of the Winter Simulation Conference*., NJ, pp. 553–560.

Page, E.H., Briggs, R. and Tufarolo, J.A. (2004) “Toward a Family of Maturity Models for the Simulation Interconnection Problem,” *In Proceedings of the Spring Simulation Interoperability Workshop*, paper 045.

Pawlaszczyk, D. and Strassburger, S. (2009) “Scalability in Distributed Simulations of Agent-Based Models.” In *Proceedings of the 41st Winter Simulation Conference*, Edited by M.D. Rossetti, R.R. Hill, B. Johansson, A. Dunkin and R.G. Ingalls, Austin, TX, December 13-16, pp. 1189-1200, Pedrielli, G., Sacco, M., Terkaj, W. and Tolio, T. (2012) “An HLA-based distributed simulation for networked manufacturing systems analysis”. *In Journal of Simulation*, Vol. 6, No.4, pp.237-252.

Petty, M.D. (2002) “Interoperability and Composability,” M&S Curriculum of Old Dominion University: Short Course Presentation, Old Dominion University, Norfolk, Virginia, (Accessed 19 March. 2015).

Petty, M.D. and Weisel, E.W. (2003) “A composability lexicon.” *In Proceedings of the 2003 Spring Simulation Interoperability Workshop*. Orlando, FL. Schriber, T. (1974) Simulation Using GPSS. New York: Wiley, pp 181-187.

Petty, M. D. and Weisel, E. W. (2004) "A theory of simulation composability," Virginia Modeling Analysis & Simulation Center, Old Dominion University, Norfolk, Virginia, (Accessed 19 March. 2015).

Petty,M.D. (2009) " Principles of Modeling and Simulation: A Multidisciplinary Approach”, Verification and Validation, John Wiley & Sons, ISBN: 978-0-470-28943-3.

Petty, M.D., Morse, K.L., Riggs, W.C., Gustavson, P. and Rutherford, H. (2010) “A reuse lexicon: terms, units, and modes in M&S asset reuse”. *In Proceedings of the Fall 2010 Simulation Interoperability Workshop*. Orlando, FL.

Pidd, M. (1992) ‘‘Object orientation and three phase simulation’’. *In Proceedings of the 1992 Winter Simulation Conference*, edited by Swain, J.J., Goldsman, D., Crain, R.C. and Wilson, J.R., Piscataway, NJ, IEEE, pp. 689–693.

Pidd, M. (2002) "Simulation Software and Model Reuse: A Polemic" *In Winter Simulation Conference*, edited by E. Yücesan, C. H. Chen, J. L. Snowdon and J. M. Charnes, San Diego, California, USA, Association for Computing Machinery Press, pp. 772-775.

Pidd, M. (2003) “Tools for Thinking. Modelling in Management Science”, Chichester, UK: John Wiley and Sons, ISBN: 978-0-471-96455-1

Pidd, M. (2006) "*Computer simulation in management science"*. 5th Edition, Hoboken, NJ: Wiley. ISBN-10: 0470092300.

Pidd, M. (2010) "Why Modelling and Model Use Matter." *In Journal of the Operational Research Society,* Vol. 61, No.1, pp.14-24.

Pooch, U. and Wall, J. (1993) "Discrete event simulation." Boca Raton, Fla., CRC Press. ISBN: 0849371740, (Accessed 18 Jul. 2013).

Portico. (2009a) "Portico Developer Documentation", volume 25. Portico, revision 1.1rc2 edition, June 2009, (Accessed 18 Jul. 2013).

Portico (2009b) "Portico User Documentation", volume 25. Portico, revision 1.1rc2 edition, June 2009, (Accessed 18 Jul. 2013).

Rainey, L.B. and Talk, A. (2015) "Modeling and Simulation Support for System of Systems Engineering Applications." Publisher: Wiley-Blackwell; 1 edition, ISBN-13: 978-1-118-46031-3.

Reese, R. and Wyatt, D. L. (1987) "Software Reuse and Simulation" In Winter Simulation Conference, edited by A. Thesen, W. Grant and W. Kelton, Association for Computing Machinery Press, pp. 185-192.

Richter, H. and Marz, L. (2000) "Toward a standard process: the use of UML for designing simulation models," *In Proceedings of 2000 Winter Simulation Conference,* Orlando, FL, Vol.1, pp. 394-398.

Robert G.B., David C.B., Paul F.R. and Joseph C.C. (2004) "J.C: In Search of the Philosopher's Stone: Simulation Composability Versus Component-Based Software Design," in *Proceedings of the Fall Simulation Interoperability Workshop,* Orlando, FL.

Robinson, S. (2008a) "Conceptual Modelling for Simulation Part I: Definition and Requirements" *In Journal of the Operational Research Society*, Vol.59, No.3, pp. 278-290.

Robinson, S. (2011) "Conceptual Modeling for Simulation." *In Wiley Encyclopedia of Operations Research and Management Science*, Edited by J.J. Cochran, forthcoming. New York: Wiley.

Robinson, S. (2013) "Conceptual Modeling for Simulation." In *Proceedings of the 2013 Winter Simulation Conference*, Washington, DC, USA, pp. 377-388.

Robinson, S. (2014) "Simulation: The Practice of Model Development and Use", Second edition, Basingstoke: Palgrave Macmillan, ISBN: 978-0-470-09278-1.

Robinson, S. (2015) "A tutorial on conceptual modeling for simulation," 2015 Winter Simulation Conference (WSC), Huntington Beach, CA, 2015, pp. 1820-1834.

Rutherford, M., Carzaniga, A., and Wolf, A. L., "Simulation-Based Testing of Distributed Systems ; CU-CS-1004-06" (2006). *Computer Science Technical Reports*. 938.

Santos, A., Leal K. and Chiroque, L. F. (2013) "Building an HLA-Based Distributed Simulation: A Metadata Approach," *In 2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications*, Delft, 2013, pp. 153-160.

Saunders, M., Lewis, P. and Thornhill, A. (2007) "Research Methods for Business Students." 5th edition. Harlow, England: Pearson Education. ISBN: 978-0-273-71686-0.

Serna,M., Sevillano,F., Beltran,M. and Guzman,A. (2010) "Defining the entity transfer interoperability reference model for military applications", *In Proceedings of the 2010 Spring Simulation Multi conference*, Orlando, Florida — April 11 – 15, No. 21,

Shannon, R.E. (1975) "System simulation: The Art and Science". Prentice-Hall, Englewood Cliffs, NJ, ISBN: 978-0-138-81839-5.

Shannon, R.E. (1998) “Introduction to the Art and Science of simulation.” *In Proceedings of the 1998 Winter Simulation Conference*, Edited by Medeiro, D.J., Waston, E.F., Carson, J.S. and Manivannan, M.S., IEEE, Piscataway, NJ, pp. 7-14.

Silva, P.M.S. and Pinto, L.R. (2010) “Emergency Medical Systems Analysis by Simulation and Optimization.” In Proceedings of the 42nd Winter Simulation Conference, Edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan and E. Yucesan, pp. 2422-2432, Baltimore, MD, December 05-08.

Strassburger, S., Schulze, T., Klein, U. and Henroksen, J. O. (1998) “Internet based simulation using off-the-shelf simulation tools and HLA”. *In Proceedings of the 30th Winter Simulation Conference*, ACM Press, New York, NY, pp. 1669-1676.

Straßburger, S. (2001) "Distributed Simulation Based on the High Level Architecture in Civilian Application Domains", PhD thesis, University Otto-von-Guericke, Magdeburg, Germany.

Sturroch, D.T. (2012) “Tutorial: Tips for Successful Practice of Simulation”. *In Proceedings of the 2012 Winter Simulation conference, Berlin*, edited by C. Laroque, J, Himmelspach, R. Pasupathy, O. Rose, and A.M. Uhrmacher, IEEE Piscataway, NJ, pp. 1-8.

Taylor, S.J.E., Bruzzone, A., Fujimoto, R., Gan, B.P., Strassburger, S. and Paul, R. J. (2002) "Distributed simulation and industry: potentials and pitfalls," *In Proceedings of the Winter Simulation Conference*, San Diego, CA, USA, vol.1, pp. 688-694.

Taylor, S.J.E., Sudra, R., Janahan, T., Tan, G. and Ladbrook, J. (2002b) “GRIDS-SCF: An Infrastructure for Distributed Supply Chain Simulation.” *Simulation*, Vol. 78, No. 5, pp. 312-320.

Taylor, S.J.E., Strassburger, S., Turner, S., Low, M., Wang, X. and Ladbrook, J. (2006) "Developing Interoperability Standards for Distributed Simulation and COTS Simulation Packages with the CSPI PDG". *In Proceedings of the 2006 Winter Simulation Conference.* Monterey, CA, pp. 1101-1110.

Taylor, S.J.E., Wang, X.G., Turner, S. J. and Low, M.Y.H (2006a) "Integrating heterogeneous distributed COTS discrete-event simulation packages: an emerging standards-based approach". *In IEEE Transactions on Systems, Man, and Cybernetics* - Part A: Systems and Humans, Vol. 36, No. 1, pp.109-122.

Taylor, S.J.E, Mustafee,N., Strassburger,S., Turner,S.J., Low,M.Y.H. and Ladbrook,J. (2007) "The SISO CSPI PDG standard for commercial off-the-shelf simulation package interoperability reference models", *In the Proceedings of the 2007 Winter Simulation Conference*, Washington, December 2007. pp. 594-602.

Taylor, S.J.E., Turner S.J., Strassburger, S. and Mustaffee, N. (2012a) "Bridging The Gap: A Standards-Based Approach to OR/MS Distributed Simulation." *ACM Transactions on Modeling and Computer Simulation*, Vol. 22, No. 4, Article 18.

Taylor, S.J.E., Fishwick, P.A., Fijimoto, R., Uhrmacher, A.M., Page, E.H. and Wainer, G. (2012b) "Panel on Grand Challenges for Modeling and Simulation." In *Proceedings of the 44th Winter Simulation Conference*, Edited by C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose and A.M. Uhrmacher, Berlin, DE, December 09- 12, pp. 2614-2628.

Taylor, S.J.E., Brailsford, S., Chick, S.E., L'Ecuyer, P., Macal, C.M. and Nelson, B.L. (2013) "Modeling and Simulation Grand Challenges: An OR/MS Perspective." In Proceedings of the 45th Winter Simulation Conference, December 08-11, pp. 1269-1282.

Taylor, S.J.E., Revagar, N., Chambers, J., Yero, M., Anagnostou, A., Nouman, A., Chaudhry, N.R. and Elfrey, P.R. (2014) "Simulation Exploration Experience: A Distributed Hybrid Simulation of a Lunar Mining Operation", *In Proceedings of* *the 2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications (DS-RT '14),* IEEE Computer Society, Washington, DC, USA, pp. 107–112.

Taylor, S.J.E, Khan, A., Morse,K.L.,Tolk,A., Yilmaz,L., Zander,J., Mosterman, P.J. (2015) "Grand Challenges for Modeling and Simulation: Simulation everywhere–from cyberinfrastructure to clouds to citizens". *In Journal of simulation*, Vol. 91, No. 7, pp. 648-665.

Tocher K.D. (1963) "The art of simulation". English University Press, London, ISBN: 978-0-340-11452-0.

Torn, A.A. (1981) "Simulation graphs: A general tool for modeling simulation designs." *In journal of Simulation,* Vol. 37, No.6, pp.187-194.

Tolk A, Muguira JA. (2003) "The levels of conceptual interoperability model". *In Fall Simulation Interoperability Workshop*. Orlando, Florida: Simulation Interoperability Standards Organization, 2003.

Tolk, A., Diallo, S. Y. and Turnitsa, C. D. (2007) "Applying the levels of conceptual interoperability model in support of integratability, interoperability, and composability for system-of-systems engineering". Faculty Bibliography. Paper 35.

Tolk, A. (2010). "Engineering Management Challenges for Applying Simulation as a Green Technology". *In Proceedings of 31st Annual National Conference of the American Society for Engineering Management*, ASEM 2010.

Tolk, A. (2013) "Interoperability, Composability, and Their Implications for Distributed Simulation: Towards Mathematical Foundations of Simulation Interoperability," *In 2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications,* Delft, pp. 3-9.

Van der Zee, D-J. (2007) "Developing Participative Simulation Models: Framing Decomposition Principles for Joint Understanding". *In Journal of Simulation,* Vol. 1, No.3, pp. 187-202.

Ulgen, O.M., Black, J.J., Johnsonbaugh, B. and Klungle, R. (1994) "Simulation Methodology – A Practitioner's Perspective." *In International Journal of Industrial Engineering,* Applications and Practice, Vol. 1, No. 2.

Vangheluwe, H., and De Lara, J. (2002) "Meta-Models are Models too" I*n Winter Simulation Conference*, edited by. E. Yücesan, C.H. Chen, J. L. Snowdon and J. M. Charnes, San Diego, California, USA: Association for Computing Machinery Press, pp. 597-605.

Vasilecas, O., Caplinskas, A. and Wojtkowski, G. (2010) "Information Systems Development: Advances in Theory, Practice, and Education". Springer US; reprint of hardcover 1st ed. 2005 edition. ISBN: 978-1-441-93768-1

Verbraeck, A., Saanen, Y., Stojanovic, Z., Shishkov, B., Meijer, A., Valentin, E., and Van der Meer, K. (2002) " What are Building Blocks?" in Building blocks for Effective Telematics Application Development and Evaluation, eds. A. Verbraeck and A. Dahanayake, Delft, The Netherlands, pp. 8-21.

Wang, X., Turner, S.J. and Taylor, S. J. E. (2006) "COTS Simulation Package (CSP) Interoperability -A Solution to Synchronous Entity Passing," 20th Workshop on Principles of Advanced and Distributed Simulation (PADS'06), Singapore, 2006, pp. 201-210.

Wang, W.G., Tolk, A. and Wang, W.P. (2009) "The Levels of Conceptual Interoperability Model: Applying Systems Engineering Principles to Modeling and Simulation." *In Proceedings of the Spring Simulation Multiconference, Society for Modeling & Simulation International (SCS)*: San Diego, CA, March 22-27. Article 168.

Weick, K.E. (1984) "Theoretical assumptions and research methodology selection." *In The Information Systems Research Challenge: Proceedings,* *Boston, Harvard Business School Press*, Edited by McFarlan, F.W., pp: 111-132.

Weisel, E.W., Mielke, R.R. and Petty, M.D. (2003) "Validity of models and classes of models in semantic composability". *In Proceedings of the Fall 2003 Simulation Interoperability Workshop*. Orlando, FL, September 14-19, pp. 14-19.

Willemain, T. R. (1995) "Model Formulation: What Experts Think About and When." *Operations Research,* Vol. 43, No.6, pp. 916-932.

Yin, R.K. (2009) "Case Study Research, Design and Methods." 4th edition, Sage Publications, CA. ISBN 978-1-4129-6099-1.

Zeigler, B. P. (1976) "Theory of Modelling and Simulation". Wiley & Sons, Incorporated, John, ISBN: 978-0-471-98152-7.

Zeigler B. (1986) "Toward a simulation methodology for variable structure modeling". *In Modeling and Simulation Methodology in the Artificial Intelligence Era*, North Holland, edited by Elzas, Oren, Zeigler, Elsevier Sci. Pub. B. V., Amsterdam, The Netherlands, pp. 195-210.

Zeigler, B. P., Praehofer, H., and Kim, T. G. (2000) "Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems", San Diego, USA: Academic Press. ISBN-13: 978-0127784557.

Appendix A

Example Discrete Event Model in Repast

```
package DiscreteEventSim;
import hla.rti1516e.exceptions.RTIexception;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import cern.jet.random.Exponential;
import cern.jet.random.Normal;
import repast.simphony.context.Context;
import
repast.simphony.context.space.continuous.ContinuousSpaceFactoryFinder;
import repast.simphony.context.space.grid.GridFactoryFinder;
import repast.simphony.dataLoader.ContextBuilder;
import repast.simphony.engine.environment.RunEnvironment;
import repast.simphony.engine.schedule.ISchedulableAction;
import repast.simphony.engine.schedule.ISchedule;
import repast.simphony.engine.schedule.ScheduleParameters;
import repast.simphony.parameter.Parameters;
import repast.simphony.random.RandomHelper;
import repast.simphony.space.continuous.ContinuousSpace;
import repast.simphony.space.continuous.RandomCartesianAdder;
import repast.simphony.space.continuous.StrictBorders;
import repast.simphony.space.grid.Grid;
import repast.simphony.space.grid.GridBuilderParameters;
import repast.simphony.space.grid.SimpleGridAdder;

/**
 * Toolkit for discrete event simulation using Repast Java.
 *
 * The model in this example is an M/M/1 queue, which can easily be
modified
 * to a G/G/s queue with some minor adjustments.  More complex systems
can be
 * modeled with the addition of new methods.
 *
 * @author Tim Sweda
 *
 */

public class DESimBuilder implements ContextBuilder<Object> {

	// Declare schedule and actions
	static ISchedule schedule;
	ISchedulableAction nextArrival, nextDeparture, nextClear,
nextEnd;


	// Declare queues, resources, random number generators, and
statistics
	Queue mm1q;
```

```
Resource mm1r;
Normal arrivalRNG;
Normal serviceRNG;
int arrivals, departures;
Stat arriveStat, departStat, lengthStat, waitStat, utilStat;

public static final String CONTEXT_ID = "ProducerModel";
public static final String SPACE_ID = "space";
public static final String GRID_ID = "grid";
public static final int GRID_SIZE_X = 60;
public static final int GRID_SIZE_Y = 40;
public static final double NEXT_TIME_JUMP = 1.0;
public static final String FEDERATENAME="Federate 1";

// Declare queue, resource, and stat lists
static List<Queue> qList;
static List<Resource> rList;
static List<Stat> sList;

// Declare system parameters
double lambda; // Mean arrival rate
double mu; // Mean service time
long serviceTime;

// Declare run parameters
int replications; // Number of replications
int repCounter;
double warmup; // Warmup time for each replication
double endTime; // Length of each replication


Parameters params;
static Federate f;
static ContinuousSpace<Object> space;
static Grid<Object> grid;
static Context<Object> maincontext;



@Override
public Context<Object> build(Context<Object> context) {

        // Initialize everything
        schedule = RunEnvironment.getInstance().getCurrentSchedule();

        params = RunEnvironment.getInstance().getParameters();
        replications = (Integer)params.getValue("replications");
        warmup = (Double)params.getValue("warmup");
        endTime = (Double)params.getValue("endTime");
        lambda = (Double)params.getValue("lambda");
        mu = (Double)params.getValue("mu");
        arrivalRNG = RandomHelper.createNormal(4,1.5);
        serviceRNG = RandomHelper.createNormal(4,1.5);

        qList = new ArrayList<Queue>();
        rList = new ArrayList<Resource>();
```

```
		sList = new ArrayList<Stat>();
		mm1q = new Queue("FIFO");
		mm1r = new Resource(1);
		// Global statistics (not part of slist)
		arriveStat = new Stat();
		departStat = new Stat();
		lengthStat = new Stat();
		waitStat = new Stat();
		utilStat = new Stat();

		repCounter = 0; // Keep track of current replication

		space = ContinuousSpaceFactoryFinder
				.createContinuousSpaceFactory(null) // No
hints
				.createContinuousSpace(SPACE_ID, context,
						new
RandomCartesianAdder<Object>(),
						new StrictBorders(),
GRID_SIZE_X, GRID_SIZE_Y);
		// Create a grid on which agents are located at
		grid = GridFactoryFinder.createGridFactory(null)
				.createGrid(GRID_ID, context,
						new
GridBuilderParameters<Object>(
						new
repast.simphony.space.grid.StrictBorders(),
						new SimpleGridAdder<Object>(),
								// Each cell in the
grid is multi-occupancy
								true,
								GRID_SIZE_X,
GRID_SIZE_Y));

		// Schedule first event

		try
		{

			f = new Federate(FEDERATENAME,this);
			context.add(f);
		}

		catch( RTIexception rtie )
		{
			// an exception occurred, just log the information
and exit

			rtie.printStackTrace();
		}
		catch( Exception e )
		{
			// an exception occurred, just log the information
and exit

			e.printStackTrace();
		}
		schedule.schedule(ScheduleParameters.createOneTime(0),
this, "initialize");

		return context;
	}

	// Initialize system before each replication
	public void initialize() {
		repCounter++;
		arrivals = 0;
		departures = 0;
		for (Queue q:qList)
			q.clear();
		for (Resource r:rList)
			r.reset();
		for (Stat s:sList)
			s.initialize();

		long firstArrival =
Math.round(schedule.getTickCount()+nxtArv());
		nextArrival =
schedule.schedule(ScheduleParameters.createOneTime(firstArrival, 1),
this, "arrive");
		nextClear =
schedule.schedule(ScheduleParameters.createOneTime(schedule.getTickCount
()+warmup, ScheduleParameters.LAST_PRIORITY), this, "clearStats");
		nextEnd =
schedule.schedule(ScheduleParameters.createOneTime(schedule.getTickCount
()+endTime, ScheduleParameters.LAST_PRIORITY), this, "end");


	}

	// Arrival event
	public void arrive() {
		arrivals++;
				Entity e = new Entity();
		if (mm1r.isAvailable(1)) {
			mm1q.skip();
			mm1r.seize(1);
			//double departTime =
schedule.getTickCount()+serviceRNG.nextDouble();
			long departTime = (long)
(schedule.getTickCount()+nxtSrv());
			if (departTime < nextEnd.getNextTime()) // This
conditional (and others like it) can be removed once removeAction is
implemented
				nextDeparture =
schedule.schedule(ScheduleParameters.createOneTime(departTime, 1), this,
"depart", e);


		}
		else
			mm1q.enqueue(e);
```

```
                long arriveTime =
Math.round(schedule.getTickCount()+nxtArv());
                System.out.println("Entity Arrived at," +
schedule.getTickCount());
                if (arriveTime < nextEnd.getNextTime())
                        nextArrival =
schedule.schedule(ScheduleParameters.createOneTime(arriveTime, 1), this,
"arrive");


        }

        // End of service event
        public void depart(Entity e) {
                System.out.println("Entity Departed at," +
schedule.getTickCount());
                Entity next = mm1q.pop();
                if (next != null) {
                        //double departTime =
schedule.getTickCount()+serviceRNG.nextDouble();
                        long departTime = (long)
(schedule.getTickCount()+nxtSrv());
                        if (departTime < nextEnd.getNextTime())
                                nextDeparture =
schedule.schedule(ScheduleParameters.createOneTime(departTime, 1), this,
"depart", next);
                }
                else
                        mm1r.release(1);
                departures++;
        }

       // Clear statistics after warmup period
       public void clearStats() {
              for (Stat s:sList)
                     s.initialize();
              arrivals = 0;
              departures = 0;
       }

       // Record and print statistics after each replication; reset
system for next replication
       public void end() {
              recordGlobalStats();
              printStats();
              if (repCounter < replications)
                     initialize();
              else {
                     System.out.println("Mean arrival rate:
"+arriveStat.getAverage()/(endTime-warmup));
                     System.out.println("Mean service rate:
"+departStat.getAverage()/(endTime-warmup));
                     System.out.println("Mean queue length:
"+lengthStat.getAverage());
                     System.out.println("Mean waiting time in queue:
"+waitStat.getAverage());
```

```
                        System.out.println("Mean resource utilization: 
"+utilStat.getAverage());
                }
        }

        // Save statistics from current replication
        public void recordGlobalStats() {
                arriveStat.recordDT(arrivals);
                departStat.recordDT(departures);
                lengthStat.recordDT(sList.get(0).getAverage());
                waitStat.recordDT(sList.get(1).getAverage());
                utilStat.recordDT(sList.get(2).getAverage());
        }

        // Print statistics to console as comma-separated list
        public void printStats() {
                System.out.print(arrivals+","+departures);
                for (Stat s:sList)
                        System.out.print(","+s.getAverage());
                System.out.println();
        }

        private long nxtArv() {
                long arrivaltime = 
Math.round(Math.abs(arrivalRNG.nextDouble()));
                if (arrivaltime < 1)
                        arrivaltime =1;
                return arrivaltime;

        }
        private long nxtSrv() {
                long servicetime = 
Math.round(Math.abs(serviceRNG.nextDouble()));
                if (servicetime < 1)
                        servicetime =1;
                return servicetime;
        }
```

Appendix B

Example federate joining and time advance

```
package DiscreteEventSim;

import hla.rti1516e.AttributeHandle;
import hla.rti1516e.AttributeHandleSet;
import hla.rti1516e.AttributeHandleValueMap;
import hla.rti1516e.CallbackModel;
import hla.rti1516e.InteractionClassHandle;
import hla.rti1516e.LogicalTime;
import hla.rti1516e.ObjectClassHandle;
import hla.rti1516e.ObjectInstanceHandle;
import hla.rti1516e.ParameterHandle;
import hla.rti1516e.ParameterHandleValueMap;
import hla.rti1516e.RTIambassador;
import hla.rti1516e.ResignAction;
import hla.rti1516e.RtiFactoryFactory;
import hla.rti1516e.encoding.EncoderFactory;
import hla.rti1516e.encoding.HLAASCIIstring;
import hla.rti1516e.encoding.HLAfloat32BE;
import hla.rti1516e.encoding.HLAfloat64BE;
import hla.rti1516e.encoding.HLAinteger16BE;
import hla.rti1516e.encoding.HLAinteger32BE;
import hla.rti1516e.encoding.HLAinteger64BE;
import hla.rti1516e.exceptions.FederatesCurrentlyJoined;
import hla.rti1516e.exceptions.FederationExecutionAlreadyExists;
import hla.rti1516e.exceptions.FederationExecutionDoesNotExist;
import hla.rti1516e.exceptions.RTIexception;
import hla.rti1516e.time.HLAfloat64Interval;
import hla.rti1516e.time.HLAfloat64Time;
import hla.rti1516e.time.HLAfloat64TimeFactory;
import java.io.BufferedReader;
import java.io.File;
import java.io.InputStreamReader;
import java.util.Random;
import java.net.MalformedURLException;
import java.net.URI;
import java.net.URL;
import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;
import java.util.List;
import repast.simphony.context.Context;
import repast.simphony.engine.environment.DefaultRunEnvironmentBuilder;
import repast.simphony.engine.environment.RunEnvironment;
import repast.simphony.engine.environment.RunListener;
import repast.simphony.engine.environment.Runner;
import repast.simphony.engine.schedule.ISchedulableAction;
import repast.simphony.engine.schedule.Schedule;
import repast.simphony.engine.schedule.ScheduleParameters;
import repast.simphony.engine.schedule.ScheduledMethod;
import repast.simphony.engine.watcher.Watch;
import repast.simphony.engine.watcher.WatcherTriggerSchedule;
import repast.simphony.essentials.RepastEssentials;
import repast.simphony.parameter.Parameters;
import repast.simphony.relogo.Utility;
import repast.simphony.space.continuous.ContinuousSpace;
import repast.simphony.space.grid.Grid;
import repast.simphony.ui.GUIScheduleRunner;


public class Federate
{


	public static final String READY_TO_RUN = "ReadyToRun";
	private ObjectInstanceHandle objectHandle;
	private double timestep = 1.0; //time increment/jump used by RTI
	private RTIambassador rtiamb;
	private FederateAmbassador fedamb;  // created when we connect
	private HLAfloat64TimeFactory timeFactory; // set when we join
	protected EncoderFactory encoderFactory;     // set when we join
	// caches of handle types - set once we join a federation
	protected ObjectClassHandle classHandle;
	protected AttributeHandle aaHandle;
	protected AttributeHandle abHandle;
	protected ParameterHandle P1Handle;

	protected InteractionClassHandle servedHandle;

	ISchedulableAction  nextDeparture;

	public Federate( String federateName,DESimBuilder db) throws
Exception
	{

		///////////////////////////////////////////////////
		// 1 & 2. create the RTIambassador and Connect //
		///////////////////////////////////////////////////
		log( "Creating RTIambassador" );
		rtiamb =
RtiFactoryFactory.getRtiFactory().getRtiAmbassador();
		encoderFactory =
RtiFactoryFactory.getRtiFactory().getEncoderFactory();


		// connect
		log( "Connecting..." );
		fedamb = new FederateAmbassador(this);
		rtiamb.connect( fedamb, CallbackModel.HLA_EVOKED );

		//////////////////////////////
		// 3. create the federation //
		//////////////////////////////

		log( "Creating Federation..." );
		// We attempt to create a new federation with the pc.xml
		try
		{
			URL[] modules = new URL[]{
			    (new File("pc.xml")).toURI().toURL()
			};

			rtiamb.createFederationExecution( "IRMTYPEA", 
modules );
			log( "Created Federation" );
		}
		catch( FederationExecutionAlreadyExists exists )
		{
			log( "Didn't create federation, it already existed" 
);
		}
		catch( MalformedURLException urle )
		{
			log( "Exception loading one of the FOM modules from 
disk: " + urle.getMessage() );
			urle.printStackTrace();
			return;
		}

		////////////////////////////
		//	4. join the federation //
		////////////////////////////
		URL[] joinModules = new URL[]{	(new 
File("pc.xml")).toURI().toURL()  };

		rtiamb.joinFederationExecution( federateName,            // 
name for the federate
		                                "IRMTYPEATYPE",   // federate type
		                                "IRMTYPEA",     // name of federation
		                                joinModules );          // modules we want to add

		log( "Joined Federation as " + federateName );

		// cache the time factory for easy access
		this.timeFactory = 
(HLAfloat64TimeFactory)rtiamb.getTimeFactory();

		////////////////////////////////
		// 5. announce the sync point //
		////////////////////////////////
		// announce a sync point to get everyone on the same page. 
if the point
		// has already been registered, we'll get a callback saying 
it failed,
		// but we don't care about that, as long as someone 
registered it
```

```
		rtiamb.registerFederationSynchronizationPoint(
READY_TO_RUN, null );
		// wait until the point is announced
		while( fedamb.isAnnounced == false )
		{
			rtiamb.evokeMultipleCallbacks( 0.1, 0.2 );
		}

		// WAIT FOR USER TO KICK US OFF
		// So that there is time to add other federates, we will
wait until the
		// user hits enter before proceeding. That was, you have
time to start
		// other federates.
		waitForUser();

		///////////////////////////////////////////////////////
		// 6. achieve the point and wait for synchronization //
		///////////////////////////////////////////////////////
		// tell the RTI we are ready to move past the sync point
and then wait
		// until the federation has synchronized on
		rtiamb.synchronizationPointAchieved( READY_TO_RUN );
		log( "Achieved sync point: " +READY_TO_RUN+ ", waiting for
federation..." );
		while( fedamb.isReadyToRun == false )
		{
			rtiamb.evokeMultipleCallbacks( 0.1, 0.2 );
		}

		/////////////////////////////
		// 7. enable time policies //
		/////////////////////////////
		// in this section we enable/disable all time policies
		// note that this step is optional!
		enableTimePolicy();
		log( "Time Policy Enabled" );

		//////////////////////////////
		// 8. publish and subscribe //
		//////////////////////////////
		// in this section we tell the RTI of all the data we are
going to
		// produce, and all the data we want to know about
		publishAndSubscribe();
		log( "Published and Subscribed" );

		/////////////////////////////////////
		// 9. register an object to update //
		/////////////////////////////////////
		this.objectHandle = registerObject();
		log( "Registered Object, handle=" + this.objectHandle );
		rtiamb.enableAsynchronousDelivery();

	}
	/**
	 * This is just a helper method to make sure all logging it
output in the same form
	 */
	private void log(String message)
	{
		System.out.println( "sender Federate   : " + message );
	}

	/**
	 * This method will block until the user presses enter
	 */
	private void waitForUser()
	{
		log( " >>>>>>>>>> Press Enter to Continue <<<<<<<<<<" );
		BufferedReader reader = new BufferedReader( new
InputStreamReader(System.in) );
		try
		{
			reader.readLine();
		}
		catch( Exception e )
		{
			log( "Error while waiting for user input: " +
e.getMessage() );
			e.printStackTrace();
		}
	}


	public void finalize()  throws RTIexception//Destructor function
	{

		///////////////////////////////////////
		// 11. delete the object we created //
		///////////////////////////////////////
		deleteObject( objectHandle );
		log( "Deleted Object, handle=" + objectHandle );

		/////////////////////////////////////
		// 12. resign from the federation //
		/////////////////////////////////////
		rtiamb.resignFederationExecution(
ResignAction.DELETE_OBJECTS );
		log( "Resigned from Federation" );

		////////////////////////////////////////
		// 13. try and destroy the federation //
		////////////////////////////////////////
		// NOTE: we won't die if we can't do this because other
federates
		//       remain. in that case we'll leave it for them to
clean up
```

```
			try
			{
				rtiamb.destroyFederationExecution( "IRMTYPEA" );
				log( "Destroyed Federation" );
			}
			catch( FederationExecutionDoesNotExist dne )
			{
				log( "No need to destroy federation, it doesn't
exist" );
			}
			catch( FederatesCurrentlyJoined fcj )
			{
				log( "Didn't destroy federation, federates still
joined" );
			}
		}


		private void enableTimePolicy() throws RTIexception
		{
			HLAfloat64Interval lookahead = timeFactory.makeInterval(
fedamb.federateLookahead );

			////////////////////////////
			// enable time constrained //
			////////////////////////////
			this.rtiamb.enableTimeConstrained();

			// tick until we get the callback
			while( fedamb.isConstrained == false )
			{
				rtiamb.evokeMultipleCallbacks( 0.1, 0.2 );
			}
		}

		private void publishAndSubscribe() throws RTIexception
		{
			this.classHandle = rtiamb.getObjectClassHandle(
"HLAobjectRoot.Producer1" );
			this.aaHandle = rtiamb.getAttributeHandle( classHandle,
"QtyEntity" );

			// package the information into a handle set
			AttributeHandleSet attributes =
rtiamb.getAttributeHandleSetFactory().create();
```

```
			try
			{
				rtiamb.destroyFederationExecution( "IRMTYPEA" );
				log( "Destroyed Federation" );
			}
			catch( FederationExecutionDoesNotExist dne )
			{
				log( "No need to destroy federation, it doesn't
exist" );
			}
			catch( FederatesCurrentlyJoined fcj )
			{
				log( "Didn't destroy federation, federates still
joined" );
			}
		}


		private void enableTimePolicy() throws RTIexception
		{
			HLAfloat64Interval lookahead = timeFactory.makeInterval(
fedamb.federateLookahead );

			/////////////////////////////
			// enable time constrained //
			/////////////////////////////
			this.rtiamb.enableTimeConstrained();

			// tick until we get the callback
			while( fedamb.isConstrained == false )
			{
				rtiamb.evokeMultipleCallbacks( 0.1, 0.2 );
			}
		}

		private void publishAndSubscribe() throws RTIexception
		{
			this.classHandle = rtiamb.getObjectClassHandle(
"HLAobjectRoot.Producer1" );
			this.aaHandle = rtiamb.getAttributeHandle( classHandle,
"QtyEntity" );

			// package the information into a handle set
			AttributeHandleSet attributes =
rtiamb.getAttributeHandleSetFactory().create();
```

```
                attributes.add( aaHandle );

                // do the actual publication
                rtiamb.publishObjectClassAttributes( classHandle,
attributes );


                ////////////////////////////////////////////////////////
                // subscribe to all attributes of ObjectRoot.Ambulance //
                ////////////////////////////////////////////////////////

                rtiamb.subscribeObjectClassAttributes( classHandle,
attributes );

                ////////////////////////////////////////////////////
                // publish the interaction class InteractionRoot.X //
                ////////////////////////////////////////////////////

                servedHandle = rtiamb.getInteractionClassHandle(
"InteractionRoot.Consumer" );
                P1Handle=rtiamb.getParameterHandle(servedHandle,
"P1QStatus");
                rtiamb.publishInteractionClass(servedHandle);

                ////////////////////////////////////////////////////////
                // subscribe to the FoodServed.DrinkServed interaction //
                ////////////////////////////////////////////////////////
                rtiamb.subscribeInteractionClass(servedHandle);
        }


        /**
         * This method will register an instance of the Soda class and
will
         * return the federation-wide unique handle for that instance.
Later in the
         * simulation, we will update the attribute values for this
instance
         */
        private ObjectInstanceHandle registerObject() throws RTIexception
        {
                return rtiamb.registerObjectInstance( classHandle );
        }

        /**
         * This method will update all the values of the given object
instance.
         * Note that we don't actually have to update all the attributes
at once, we
         * could update them individually, in groups or not at all!
         */

        public void updateAttributeValues() throws RTIexception
        {
                ///////////////////////////////////////////////
```

```
             // create the necessary container and values //
             ///////////////////////////////////////////////
             // create a new map with an initial capacity - this will
grow as required
             AttributeHandleValueMap attributes =
rtiamb.getAttributeHandleValueMapFactory().create(5);

             HLAinteger32BE aaValue =
encoderFactory.createHLAinteger32BE( 1 );

             attributes.put( aaHandle, aaValue.toByteArray() );

             //////////////////////////
             // do the actual update //
             //////////////////////////
             rtiamb.updateAttributeValues( objectHandle, attributes,
generateTag() );
             HLAfloat64Time time = timeFactory.makeTime(
fedamb.federateTime+fedamb.federateLookahead );
             rtiamb.updateAttributeValues( objectHandle, attributes,
generateTag(), time);

       }

       /**
        * This method will send out an interaction of the type
InteractionRoot.X. Any
        * federates which are subscribed to it will receive a
notification the next time
        * they tick(). Here we are passing only two of the three
parameters we could be
        * passing, but we don't actually have to pass any at all!
        */
       public void sendInteraction(int a ) throws RTIexception
       {

             //////////////////////////
             //    send the interaction //
             //////////////////////////
             ParameterHandleValueMap parameters =
rtiamb.getParameterHandleValueMapFactory().create(1);
             HLAfloat64BE timestamp =
encoderFactory.createHLAfloat64BE();
             HLAinteger32BE aaValue =
encoderFactory.createHLAinteger32BE(a);
             parameters.put(P1Handle, aaValue.toByteArray());

             rtiamb.sendInteraction( servedHandle, parameters,
generateTag() );

             HLAfloat64Time time = timeFactory.makeTime(
fedamb.federateTime+fedamb.federateLookahead );
             double timetag = (DESimBuilder.schedule.getTickCount()) +
(System.currentTimeMillis()/ (double)10000000000000.0 );
```

```
		timestamp = encoderFactory.createHLAfloat64BE(timetag);

	}

	/**
	 * This method will request a time advance to the current time,
plus the given
	 * timestep. It will then wait until a notification of the time
advance grant
	 * has been received.
	 */


	@ScheduledMethod(start = 1, interval = 1, priority =
ScheduleParameters.LAST_PRIORITY)
	public void advanceTime() throws RTIexception
	{
		fedamb.isAdvancing = true;
		Parameters params =
RunEnvironment.getInstance().getParameters();
		sendInteraction(5);
		if (RepastEssentials.GetTickCount() >=
(Double)params.getValue("endTime"))
			{
				finalize();
			}
			else
			{
				HLAfloat64Time time = timeFactory.makeTime(
fedamb.federateTime + timestep );
				rtiamb.timeAdvanceRequest( time );

				while( fedamb.isAdvancing )
				{
					rtiamb.evokeMultipleCallbacks( 0.1, 0.2
);

				}

			}

	}

	/**
	 * This method will attempt to delete the object instance of the
given
	 * handle. We can only delete objects we created, or for which we
own the
	 * privilegeToDelete attribute.
	 */
	private void deleteObject( ObjectInstanceHandle handle ) throws
RTIexception
```

```
        {
                rtiamb.deleteObjectInstance( handle, generateTag() );
        }


        private short getTimeAsShort()
        {
                return (short)fedamb.federateTime;
        }

        private byte[] generateTag()
        {
                return ("(timestamp)
"+System.currentTimeMillis()).getBytes();
        }
        public double getFedTime()
        {
                return fedamb.federateTime;
        }

}
```