FRAMEWORK FOR DEVELOPING DISTRIBUTED INTELLIGENCE SYSTEMS IN A PEER-TO-PEER ENVIRONMENT

by

Colin Chi-Wong Ng
B.A.Sc. (Hon.), Simon Fraser University, 2006

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in the
School of Engineering Science

© Colin C. W. Ng 2007

SIMON FRASER UNIVERSITY

Spring 2007

All rights reserved. This work may not be reproduced in whole or in part, by photocopy or other means, without permission of the author.
APPROVAL

Name: Colin Chi-Wong Ng
Degree: Master of Applied Science
Title of Thesis: Framework for Developing Distributed Intelligence Systems in a Peer-to-Peer Environment

Examinining Committee:

Chair: Dr. John Jones
Faculty of Applied Sciences Associate Dean for Undergraduate Studies

Dr. William A. Gruver
Senior Supervisor
Professor Emeritus of Engineering Science

Mr. Dorian Sabaz
Technical Supervisor
Chief Technical Officer of Intelligent Robotics Corporation

Dr. Ljiljana Trajkovic
Supervisor
Professor of Engineering Science

Dr. Dilip Kotak
Internal Examiner
Adjunct Professor of the School of Engineering Science

Date Defended/Approved: December 18, 2006
DECLARATION OF
PARTIAL COPYRIGHT LICENCE

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the “Institutional Repository” link of the SFU Library website <www.lib.sfu.ca> at: <http://ir.lib.sfu.ca/handle/1892/112>) and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author’s written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, BC, Canada

Revised: Fall 2006
ABSTRACT

In Distributed Intelligence Systems, holons collaborate to achieve local goals, and more importantly, system goals. Although present-day systems claim to be distributed intelligence systems, they lack the required intelligence needed to adapt to a dynamic network environment. The Holonic Logistics System (HLS) is a framework that facilitates the development of distributed intelligence systems in a peer-to-peer environment. With an infrastructure based completely on autonomous cooperating agents, the HLS is an example of a Distributed Intelligence System that incorporates the required intelligence into the framework. This thesis discusses the design and implementation of the HLS.

Keywords: agents; Distributed Intelligence Systems; holons; multi-agent systems

Subject Terms: Distributed Intelligence Systems; multi-agent systems
DEDICATION

I dedicate this thesis to my father, mother and sister. It was their constant support and encouragement that helped me finish my masters in such a short time. To my father and mother, I would like to thank you both for being patient with me and for guiding me throughout this final journey in university. The perseverance shown throughout your lives were an inspiration to me to see this degree through with the highest of standards. To my sister, I would like to thank you for always enlightening me with your humour and kindness when it seemed like obstacles were always in my project’s path.
ACKNOWLEDGEMENTS

I would like to thank Dorian Sabaz for his guidance and support throughout the course of this thesis. I am very grateful and appreciative of his continuous and unwavering aid in my development as both a software developer and a professional individual.

I would like to thank Dr. William A. Gruver for his support and administrative flexibility during this thesis.

I would like to thank Zafeer Alibhai for his cooperation and support during this project. Our complementary skill sets and our similar hunger for success have shown us what we are capable of, and more importantly, what we have to do to achieve that success.

Finally, I would also like to thank Dr. Ljiljana Trajkovic of the School of Engineering Science, Simon Fraser University, Dr. Dilip Kotak of the School of Engineering Science, Simon Fraser University, and Roger Lum for their continuous support.
# TABLE OF CONTENTS

Approval ...................................................................................................................................... ii
Abstract ....................................................................................................................................... iii
Dedication ................................................................................................................................... iv
Acknowledgements ...................................................................................................................... v
Table of Contents ........................................................................................................................ vi
List of Figures .............................................................................................................................. x
List of Listings ............................................................................................................................. xv
List of Tables ............................................................................................................................... xvii
Acronyms ...................................................................................................................................... xviii

## 1 INTRODUCTION ....................................................................................................................... 1

1.1 Centralized Networks ............................................................................................................ 1
   1.1.1 Scalability ..................................................................................................................... 2
   1.1.2 Fault Tolerance ............................................................................................................ 4
   1.1.3 Security and Privacy ..................................................................................................... 5
   1.1.4 Communications Topology ........................................................................................... 7

1.2 Decentralized Networks and their Benefits ........................................................................... 8
   1.2.1 Scalability in Decentralized Networks ......................................................................... 9
   1.2.2 Fault-Tolerance and Communications in Decentralized Networks ......................... 10
   1.2.3 Security and Privacy in Decentralized Networks ....................................................... 11
   1.2.4 Communications Topology in Decentralized Networks ............................................ 13

1.3 Shortcomings of Centralized Systems using Decentralized Networks ............................. 14
   1.3.1 Scalability in Centralized Systems .............................................................................. 14
   1.3.2 Fault Tolerance in Centralized Systems ..................................................................... 14
   1.3.3 Security and Privacy in Centralized Systems ............................................................ 15
   1.3.4 Communications in Centralized Systems .................................................................. 15

1.4 Decentralized Systems ........................................................................................................ 17

1.5 Distributed Systems ............................................................................................................. 18
   1.5.1 Present Multi-Agent Frameworks .............................................................................. 20
      1.5.1.1 Cognitive Agent Architecture (Cougaar) .............................................................. 21
      1.5.1.2 FIPA Abstract Architecture ............................................................................... 23
      1.5.1.3 Jack-in-the-Net Architecture (Ja-Net) ................................................................. 25
      1.5.1.4 Java Agent Development Framework (JADE) .................................................... 27
      1.5.1.5 JXTA ................................................................................................................... 32
   1.5.2 Shortcomings of Present Multi-Agent Frameworks .................................................... 36
1.5.2.1 Use of Centralized Containers ........................................... 36
1.5.2.2 Lack of Message Control ................................................. 36
1.5.2.3 Use of Client / Server Technology .................................... 37
1.5.2.4 Memory / Thread Management ........................................ 37
1.5.3 Present Distributed Systems .............................................. 37
1.5.3.1 BitTorrent (BT) ............................................................... 38
1.5.3.2 Gnutella ................................................................. 41
1.5.4 Shortcomings of Present Distributed Systems ......................... 44
1.5.4.1 More Cooperation and Collaboration Needed ...................... 44
1.5.4.2 More Intelligence Needed ............................................. 45
1.5.4.3 Generated Network Traffic ............................................ 45
1.5.4.4 Lack of Security ...................................................... 45
1.5.4.5 Lack of Optimized Caching .......................................... 46
1.5.4.6 Lack of Optimized Routing .......................................... 46
1.6 Scope Of Thesis .................................................................. 46
1.6.1 Objective ....................................................................... 46
1.6.2 Outline ......................................................................... 48
2 ARCHITECTURE AND DESIGN ................................................... 50
2.1 System Objectives .............................................................. 50
2.1.1 Accomplishment of a Global Task ..................................... 51
2.1.2 Adapting to a Changing Environment ................................ 54
2.1.3 Designing for Expansion ............................................... 56
2.2 Implementation Objectives ................................................... 56
2.2.1 Holonic Logistics System versus Holonic Strategic System ...... 57
2.2.2 Internal Communications versus External Communications ... 59
2.2.3 Standard Message Structure ............................................ 62
2.2.4 Agent Architecture versus Holon Architecture .................. 62
2.2.5 Holonic Application Architecture ..................................... 65
2.2.6 Management of Multiple Virtual Networks ....................... 67
2.2.7 Distribution of Network Resources and Network Intelligence ... 69
2.2.8 Dynamic Acquisition of Intelligence .................................. 70
2.3 Holonic Agents in the Holonic Technology Platform ................. 72
2.3.1 Administration Agent ..................................................... 73
2.3.2 Algorithms Agent .......................................................... 77
2.3.3 Applications Manager Agent .......................................... 78
2.3.4 Console Agent .............................................................. 82
2.3.5 Database Agent ............................................................ 83
2.3.6 Development Agent ....................................................... 86
2.3.7 Message Analyzer Agent and Messenger Agent ................ 88
2.3.8 Software Transfer Agent ................................................ 93
2.3.9 Systems Agent ............................................................. 96
2.3.10 Version Tracking Agent ............................................... 97
2.3.11 Virtual Networks Manager Agent .................................. 98
2.4 Distributed Intelligence in the HTP ........................................ 100
2.5 Summary ........................................................................ 101
| Figure 1.1 | Examples of client-server architectures | 1 |
| Figure 1.2 | Scalability in client-server architectures | 3 |
| Figure 1.3 | Fault-tolerance in client-server architectures | 4 |
| Figure 1.4 | Security and privacy in client-server architectures | 6 |
| Figure 1.5 | Lines of communication in client-server architecture | 7 |
| Figure 1.6 | Communications topology in centralized networks | 8 |
| Figure 1.7 | Example of a decentralized network | 9 |
| Figure 1.8 | Fault-tolerance and communications in decentralized networks | 11 |
| Figure 1.9 | Security and privacy in decentralized networks | 12 |
| Figure 1.10 | Communications topology in decentralized networks | 13 |
| Figure 1.11 | Communications in centralized systems | 16 |
| Figure 1.12 | Decentralized information and decentralized processing | 18 |
| Figure 1.13 | Distributed information and distributed processing | 19 |
| Figure 1.14 | Example of Cougaar node architecture | 23 |
| Figure 1.15 | Ja-Net framework | 26 |
| Figure 1.16 | Intra-platform communication in JADE | 30 |
| Figure 1.17 | Inter-platform communication in JADE | 31 |
| Figure 1.18 | Layering in JXTA platform | 35 |
| Figure 1.19 | Example of an executing BitTorrent | 40 |
| Figure 1.20 | Query requests in Gnutella | 42 |
| Figure 1.21 | Query responses in Gnutella | 43 |
| Figure 1.22 | Grading scheme comparing infrastructures and systems | 48 |
| Figure 2.1 | Scenario of nodes not working towards a global objective | 52 |
| Figure 2.2 | Scenario of nodes working towards a global objective | 52 |
| Figure 2.3 | Coordination of tasks within a holon | 54 |
| Figure 2.4 | Agent layout after addition of an agent | 55 |
| Figure 2.5 | Layers in the Holonic Technology Platform | 58 |
| Figure 2.6 | Systems within the Holonic Technology Platform | 59 |
| Figure 2.7 | Agent switchboard used for internal communications [31] | 60 |
| Figure 2.8 | Interface for external communications | 61 |
| Figure 2.9 | Message structure of external messages | 62 |
| Figure 2.10 | Relationship between holon and agents | 63 |
Figure 2.11 Generic architecture of an agent’s core .......................................................... 63
Figure 2.12 Agent packages in the Holonic Logistics System ........................................... 64
Figure 2.13 Relationship between holonic applications and holons .................................. 65
Figure 2.14 Agent packages in a holonic application ....................................................... 66
Figure 2.15 Relationship between a physical connection and a logical connection .......... 67
Figure 2.16 Examples of virtual layers ............................................................................. 68
Figure 2.17 Potential global networks ............................................................................... 69
Figure 2.18 Need for ability to acquire intelligence .......................................................... 71
Figure 2.19 Logic flow of system shutdown ..................................................................... 74
Figure 2.20 Logic flow of system initialization .................................................................. 75
Figure 2.21 Functionality of Administration Agent ........................................................... 76
Figure 2.22 Functionality of Algorithms Agent ................................................................. 78
Figure 2.23 Functionality of Applications Manager Agent ............................................... 79
Figure 2.24 Logic flow of registering a holonic application ............................................... 80
Figure 2.25 Search mechanism in applications manager agent ......................................... 81
Figure 2.26 Functionality of Console Agent ..................................................................... 82
Figure 2.27 Logic flow of showing an agent’s menu ....................................................... 83
Figure 2.28 Logic flow of system-backup ......................................................................... 84
Figure 2.29 Logic flow of system-restoration ................................................................... 85
Figure 2.30 Functionality of Database Agent .................................................................... 86
Figure 2.31 Functionality of Development Agent ............................................................. 86
Figure 2.32 Hierarchy for handling incoming external messages ....................................... 88
Figure 2.33 Logic flow of handling incoming external messages ....................................... 90
Figure 2.34 Logic flow of handling outgoing external messages ....................................... 92
Figure 2.35 Functionality of Messenger Agent and Message Analyzer Agent ................. 93
Figure 2.36 Functionality of Software Transfer Agent .................................................... 93
Figure 2.37 Logic flow of transferring holonic applications ............................................. 94
Figure 2.38 Logic flow of transferring holonic agents ..................................................... 95
Figure 2.39 Functionality of Systems Agent ..................................................................... 97
Figure 2.40 Functionality of Version Tracking Agent ...................................................... 98
Figure 2.41 Functionality of Virtual Networks Manager Agent ........................................ 99
Figure 3.1 Multiple threads working on a task .................................................................. 103
Figure 3.2 Use of threads in Holonic Logistics System .................................................... 106
Figure 3.3 Virtual bridge created by UDP ....................................................................... 110
Figure 3.4 Internal messaging mechanism ...................................................................... 112
Figure 3.5 Architecture of Messenger Agent ................................................................. 113
Figure 3.6 Logic flow of message backup mechanism .................................................... 115
Figure 3.7 Logic flow of send-reply protocol between two holons .................................. 115
| Figure 3.8 | Example of log-file of incoming external messages | 116 |
| Figure 3.9 | Example of log-file of user inputs | 117 |
| Figure 3.10 | Architecture of Database Agent | 118 |
| Figure 3.11 | Example of usage of nested snapshots | 118 |
| Figure 3.12 | Agent not in software listing | 125 |
| Figure 3.13 | Communications setup file | 126 |
| Figure 3.14 | Logic flow of updating code on a holon | 127 |
| Figure 3.15 | Logic flow of adding an agent to a holon | 135 |
| Figure 3.16 | Architecture of Algorithms Agent | 143 |
| Figure 3.17 | Logic flow of agent creation | 151 |
| Figure 3.18 | Logic flow of holonic application creation | 154 |
| Figure 3.19 | White pages vs. yellow pages | 159 |
| Figure 3.20 | Logic Flow of Displaying White and Yellow Pages | 161 |
| Figure 3.21 | Abstraction of Levels of Information Storage | 162 |
| Figure 3.22 | Relationship between holon and internal holonic systems | 164 |
| Figure 3.23 | Inter-holon communications | 165 |
| Figure 3.24 | Information traversing from HSS to HCS | 166 |
| Figure 3.25 | Information traversing from HCS to HSS | 167 |
| Figure 3.26 | Determining network size | 169 |
| Figure 3.27 | Logic flow of synchronization using atomic clock | 179 |
| Figure 3.28 | Logic flow of synchronization using messages | 180 |
| Figure 3.29 | Invocation of IA’s functionality | 181 |
| Figure 3.30 | Queue of responses and associated response mechanism | 184 |
| Figure 3.31 | Example of decision to transfer agent | 185 |
| Figure 4.1 | Launching of Holonic Logistics System | 187 |
| Figure 4.2 | Screen shot of platform launching | 188 |
| Figure 4.3 | Screen shot of launched platform | 188 |
| Figure 4.4 | Screen shot of Administration Agent menu | 189 |
| Figure 4.5 | Screen shot of System Agent menu | 190 |
| Figure 4.6 | Screen shot of synchronization with atomic clock | 191 |
| Figure 4.7 | Network topology to be created | 192 |
| Figure 4.8 | Screen shot of Virtual Networks Manager Agent menu | 193 |
| Figure 4.9 | Screen shot of system state prior to creation of VNET | 193 |
| Figure 4.10 | Screen shot of creation of VNET | 194 |
| Figure 4.11 | Screen shot of system state after creation of VNET | 194 |
| Figure 4.12 | Representation of network after creation of VNETs | 196 |
| Figure 4.13 | Screen shot of system state prior to creation of virtual links | 196 |
| Figure 4.14 | Screen shot of creation of virtual links | 197 |
Figure 4.53 Screen shot of reply message ................................................................. 223
Figure 4.54 Summary of events to transpire .............................................................. 224
Figure 4.55 Screen shot of step-through of intelligence analysis 1 ............................. 225
Figure 4.56 Screen shot of step-through of intelligence analysis 2 ............................. 226
Figure 4.57 Screen shot of step-through of intelligence analysis 3 ............................. 226
Figure 4.58 Activity screen shot of step-through of intelligence analysis 4 .................. 227
Figure 4.59 Activity screen shot of step-through of intelligence analysis 5 ................. 227
Figure 4.60 Screen shot of step-through of intelligence analysis 6 ............................. 228
Figure 4.61 Screen shot of decision to transfer agent .................................................. 228
Figure 4.62 Screen shot of details of agent to be transferred ...................................... 229
Figure 5.1 Fragmentation of network ................................................................. 236
Figure 5.2 Solution to privacy issue ............................................................................ 240
Figure 5.3 Solution to generated traffic issue ............................................................ 242
Figure 5.4 Relationship between holon and agents ..................................................... 243
Figure 5.5 Port-forwarding problem .......................................................................... 251
# LIST OF LISTINGS

<table>
<thead>
<tr>
<th>Listing</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Example of synchronized method.</td>
<td>108</td>
</tr>
<tr>
<td>3.2</td>
<td>Example of backing up of an agent.</td>
<td>122</td>
</tr>
<tr>
<td>3.3</td>
<td>Example of restoration of an agent.</td>
<td>123</td>
</tr>
<tr>
<td>3.4</td>
<td>Part 1 of code that updates agent code.</td>
<td>128</td>
</tr>
<tr>
<td>3.5</td>
<td>Part 2 of code that updates agent code.</td>
<td>129</td>
</tr>
<tr>
<td>3.6</td>
<td>Part 3 of code that updates agent code.</td>
<td>130</td>
</tr>
<tr>
<td>3.7</td>
<td>Part 4 of code that updates agent code.</td>
<td>131</td>
</tr>
<tr>
<td>3.8</td>
<td>Part 5 of code that updates agent code.</td>
<td>132</td>
</tr>
<tr>
<td>3.9</td>
<td>Part 6 of code that updates agent code.</td>
<td>133</td>
</tr>
<tr>
<td>3.10</td>
<td>Part 7 of code that updates agent code.</td>
<td>134</td>
</tr>
<tr>
<td>3.11</td>
<td>Part 1 of code that adds new agent.</td>
<td>136</td>
</tr>
<tr>
<td>3.12</td>
<td>Part 2 of code that adds new agent.</td>
<td>137</td>
</tr>
<tr>
<td>3.13</td>
<td>Part 3 of code that adds new agent.</td>
<td>138</td>
</tr>
<tr>
<td>3.14</td>
<td>Part 4 of code that adds new agent.</td>
<td>139</td>
</tr>
<tr>
<td>3.15</td>
<td>Part 5 of code that adds new agent.</td>
<td>140</td>
</tr>
<tr>
<td>3.16</td>
<td>Part 6 of code that adds new agent.</td>
<td>141</td>
</tr>
<tr>
<td>3.17</td>
<td>Part 7 of code that adds new agent.</td>
<td>142</td>
</tr>
<tr>
<td>3.18</td>
<td>Part 1 of Algorithm Class.</td>
<td>146</td>
</tr>
<tr>
<td>3.19</td>
<td>Part 2 of Algorithm Class.</td>
<td>147</td>
</tr>
<tr>
<td>3.20</td>
<td>Required item 1.</td>
<td>148</td>
</tr>
<tr>
<td>3.21</td>
<td>Required item 2.</td>
<td>149</td>
</tr>
<tr>
<td>3.22</td>
<td>Code initiating algorithm.</td>
<td>149</td>
</tr>
<tr>
<td>3.23</td>
<td>Code performing algorithm evaluation.</td>
<td>150</td>
</tr>
<tr>
<td>3.24</td>
<td>Part 1 of generated agent code.</td>
<td>152</td>
</tr>
<tr>
<td>3.25</td>
<td>Part 2 of generated agent code.</td>
<td>153</td>
</tr>
<tr>
<td>3.26</td>
<td>Part 3 of generated agent code.</td>
<td>154</td>
</tr>
<tr>
<td>3.27</td>
<td>Part 1 of code used to copy and rename a file.</td>
<td>155</td>
</tr>
<tr>
<td>3.28</td>
<td>Part 2 of code used to copy and rename a file.</td>
<td>156</td>
</tr>
<tr>
<td>3.29</td>
<td>Code Used to Run Renamed Class-File.</td>
<td>157</td>
</tr>
<tr>
<td>3.30</td>
<td>Generic Code Used To Test a Class File.</td>
<td>158</td>
</tr>
<tr>
<td>3.31</td>
<td>Part 1 of code determining maximum hop number.</td>
<td>171</td>
</tr>
<tr>
<td>3.32</td>
<td>Part 2 of code determining maximum hop number.</td>
<td>172</td>
</tr>
<tr>
<td>3.33</td>
<td>Part 3 of code determining maximum hop number.</td>
<td>173</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 2.1 Internal Communications vs. External Communications ........................................ 60
Table 2.2 Agents in the Holonic Technology Platform [33] ..................................................... 72
Table 2.3 Information sent to HLS from a holonic application ............................................. 80
Table 3.1 Delay times based on category .................................................................................... 183
Table 5.1 Agent functionality ...................................................................................................... 244
<table>
<thead>
<tr>
<th>ACRONYMS</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC</td>
<td>Agent Communication Channel</td>
</tr>
<tr>
<td>ACL</td>
<td>Agent Communication Language</td>
</tr>
<tr>
<td>AMS</td>
<td>Agent Management System</td>
</tr>
<tr>
<td>BT</td>
<td>BitTorrent</td>
</tr>
<tr>
<td>CE</td>
<td>Cyber Entitites</td>
</tr>
<tr>
<td>CMTS</td>
<td>Cougaar Message Transport Service</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>COUGAAR</td>
<td>Cognitive Agent Architecture</td>
</tr>
<tr>
<td>DF</td>
<td>Directory Facilitator</td>
</tr>
<tr>
<td>DIS</td>
<td>Distributed Intelligence System</td>
</tr>
<tr>
<td>DS</td>
<td>Distributed System</td>
</tr>
<tr>
<td>FIPA</td>
<td>Foundation for Intelligent Physical Agents</td>
</tr>
<tr>
<td>HCS</td>
<td>Holonic Communications System</td>
</tr>
<tr>
<td>HLS</td>
<td>Holonic Logistics System</td>
</tr>
<tr>
<td>HSS</td>
<td>Holonic Strategic System</td>
</tr>
<tr>
<td>HTP</td>
<td>Holonic Technology Platform</td>
</tr>
<tr>
<td>IMTP</td>
<td>Internal Message Transport Protocol</td>
</tr>
<tr>
<td>Ja-Net</td>
<td>Jack-in-the-Net Architecture</td>
</tr>
<tr>
<td>JADE</td>
<td>Java Agent Development Framework</td>
</tr>
<tr>
<td>JXTA</td>
<td>Juxtapose</td>
</tr>
<tr>
<td>MA</td>
<td>Messenger Agent</td>
</tr>
<tr>
<td>MAA</td>
<td>Message Analyzer Agent</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>VNET</td>
<td>Virtual Network</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

1.1 Centralized Networks

Centralized networks are built on the foundation of a client-server architecture. As illustrated in Figure 1.1, information in a centralized network is stored in a centralized server, and is passed from client to client via the server. The shortcomings imposed on such architectures include large storage capability, significant processing power, and reliable communications between clients and servers [1][2]. As these centralized networks grow in size, the strains placed on these resources can increase exponentially, resulting in the inability of such architectures to meet the requirements of the clients.

Regardless of how many servers are being used, the information is still stored and processed in a centralized location. Therefore, this centralized location acts as a centralized server.

Figure 1.1 Examples of client-server architectures.
The next sections will discuss the following shortcomings in centralised networks: scalability, fault-tolerance, security and privacy, and communications topology.

1.1.1 Scalability

A server has a finite storage capacity and a finite processing ability. This means that at any given time, the server is only capable of servicing a fixed number of clients. As technology inevitably progresses, with additional topological complexity and with an increase in fixed clients, the server will be strained beyond its capability and will not be able to service the network’s needs. The inability to service the network’s needs will result in either a lost or rejected access request. This suggests that a centralized network is only satisfactory when the number of clients does not increase significantly. Clients are forced to compete for access to the server when the number of clients does increase significantly, thereby causing congestion in the network. Figure 1.2 illustrates this latter difficulty.
Server is servicing near maximum capability

Server is servicing maximum capability

Server is servicing more than maximum capability

Unstable access requests to server as number of clients has created increased network congestion

Lost or denied access requests to server as number of clients has exceeded capability of server

Figure 1.2 Scalability in client-server architectures.
1.1.2 Fault Tolerance

Because all the critical information is stored in a centralized location, it is essential that the servers within the centralized location be reliable and consistent in responding to the requests from the clients. Failure of the centralized location to respond in such a manner could potentially result in a disastrous outcome in the network. The outcome may range from the abrupt halting of information exchange to the complete shutdown of the network. A centralized network must be capable of tolerating any faults within the server so to minimize such effects. However, because the centralized location is the only source of critical information and because the network traffic requesting for information is solely directed at that location, centralized networks are at a high-risk for collapsing from network faults. Figure 1.3 illustrates an example of a fault within the server resulting from hardware failure.

Figure 1.3 Fault-tolerance in client-server architectures.
In addition to losing access to critical information that is stored in the centralised server, when a fault occurs in a centralised system, clients also lose access to services provided by the server. Since the centralised server is the only service provider in the network, the network will break down because of the complete loss of services.

1.1.3 Security and Privacy

In addition to being able to tolerate failure within a centralized server, there is also a need for a centralized network to protect itself from network intruders. Storing all critical information in a centralized location indicates that all confidential information is also stored in that area of the centralized network. That information is at a high risk of being accessed by unauthorized individuals. As a result, a centralized network places a heavy burden on the security system used to safeguard the confidential data. As Figure 1.4 illustrates, if the security system fails in that one location, that confidential data is available to the intruder. Examples of businesses that utilize centralized networks are banks and investment firms.
Figure 1.4 Security and privacy in client-server architectures.
1.1.4 Communications Topology

Centralized networks use relay stations to pass information from one client to another client. Figure 1.5 shows an example of a centralized network utilizing such relay stations. The example shows that the disadvantage of using such stations is that if the stations fail, portions of or the entire network will fail. Hence, using a centralized architecture for communications is another risk that is inherent in centralized networks.

![Diagram of a centralised network with relay stations](image)

Figure 1.5 Lines of communication in client-server architecture.

In addition to the abovementioned shortcoming, a centralized communications topology also poses another problem in that lines of communications are limited, as shown in
1.2 Decentralized Networks and their Benefits

Decentralized networks offer an alternative network structure to the previously discussed centralized networks. Also known as distributed networks, decentralized networks are network topologies built on the foundation that every node on a network is both a server and a client, meaning that rather than concentrating information and processing capability in one area of the network, that same information and processing capability is spread
throughout the entire network [3] [4]. As a result, as illustrated in Figure 1.7, there is no longer a need to employ the use of centralized servers.

![Diagram of a decentralized network]

Figure 1.7 Example of a decentralized network.

The next few sections will discuss how decentralized networks address the following issues: scalability, fault-tolerance and communications, security and privacy, and communications topology.

### 1.2.1 Scalability in Decentralized Networks

Since no centralized server exists, burdens such as the storage of information and the processing of data are the responsibility of all the nodes in the network. The benefit of this is that since the storage and processing requirements are distributed throughout the
network, the burdens that would have been placed on a centralized server or centralized location will also be distributed throughout the network. This is advantageous because as the number of network nodes increases, assuming the network burdens remain the same, the requirements of each node reduces since the workload is divided amongst even more nodes. Therefore, by adding more nodes to the network, contrary to the effect suggested by a centralized network, a decentralized network experiences a more efficient ability to cope with the network’s needs. Furthermore, should the burdens placed on the network increase, by adding more nodes, the increase in the workload is shared by all the network nodes. This demonstrates that decentralized networks are capable of handling network topologies that are continuously increasing.

1.2.2 Fault-Tolerance and Communications in Decentralized Networks

As discussed in Section 1.1, because of its architecture, a centralized network has an inherent risk of a complete network-collapse in the event of a failure in the server or in the event of a failure in the relay stations found in such networks. A decentralized network overcomes these risks because of its architecture.

In the case of a server failure, given that each node in a decentralized network acts as both a server and a client, if several nodes fail, the rest of the network is still able to function properly since the other nodes are not reliant on the failing nodes. Several functions specific to the failing nodes may not be available, but functionality not specific to those nodes can still operate as expected. In the case of a relay-station failure, since
each node in a decentralized network acts as both a server and a client, the node has the innate ability to act as a relay station. If a node in the decentralized network is not functioning, its failure will be detected, and routing paths will be directed around the failing node, as demonstrated in Figure 1.8.

![Diagram of fault-tolerance and communications in decentralized networks](image)

**Figure 1.8** Fault-tolerance and communications in decentralized networks.

### 1.2.3 Security and Privacy in Decentralized Networks

Since vital information is stored throughout a decentralized network, sensitive information can also be stored throughout the network. This feature is very significant because unlike the case presented in Section 1.1.3, an unauthorized intrusion into a node on a decentralized network is not so catastrophic. When the confidential information is
distributed throughout the network, an intruder on a node only has a piece of the complete information. Furthermore, in order to make sense of the unauthorized information, the intruder would have to break the encryption that is used before and after the splitting of the sensitive information [5]. As a result, storing information in a decentralized network significantly reduces the risk of stolen confidential data, as illustrated in Figure 1.9.

Figure 1.9 Security and privacy in decentralized networks.
1.2.4 Communications Topology in Decentralized Networks

Figure 1.10 illustrates an example of a decentralized network. The wireless topology allows for multiple conversations between nodes as there is no centralized bus topology that the messages go through.

Figure 1.10 Communications topology in decentralized networks.
1.3 Shortcomings of Centralized Systems using Decentralized Networks

Presently, many software developers are deploying centralized systems on decentralized network infrastructures in an attempt to utilize the benefits offered by a decentralized network. However, a centralized system does not utilize the benefits offered by a decentralized network, simply because a centralized system has issues that are similar to the ones encountered by centralized networks. The next few sections will discuss the following shortcomings in centralised systems: scalability, fault-tolerance, security and privacy, and communications topology.

1.3.1 Scalability in Centralized Systems

Centralized systems process the information in one centralized location. This means that as the number of nodes on the network increases, the system in the centralized location will start to experience a decrease in performance as hardware resources become exhausted. Although adding hardware resources will temporarily alleviate the problem, it is not a solution as there is a finite limit on the amount of hardware resources that can be added.

1.3.2 Fault Tolerance in Centralized Systems

Since centralized systems process the information in one centralized location, if the systems used in the centralized location fail, then the entire system collapses.
1.3.3 Security and Privacy in Centralized Systems

Since centralized systems store information in one centralized location, if the location fails to protect against network intrusions, then all information will be accessible by the intruder.

1.3.4 Communications in Centralized Systems

Although the network infrastructure is decentralized, because the system is centralized, the processing will be performed in a centralized location. As a result, once the information has been processed, any need to exchange information will also be done in a centralized manner because the information will also have to be analyzed at centralized locations, as shown in Figure 1.11. The diagram shows a clear definition between the network layer and the system layer. Although there is no direct network connection between System A and the Centralized System Processor, System A and the Centralized System Processor view each other as neighbours because of the connection path offered by the network infrastructure via network layer under System C and System E. The system information must travel to the Centralized System Analyzer, demonstrating the issue of centralized communication. Because the analyses are performed in one centralized location, the system information has a set path to travel, indicating that alternative paths provided by the network infrastructure are wasted. This lack of functionality is not a network issue, but rather is a system issue.
Figure 1.11 Communications in centralized systems.
1.4 Decentralized Systems

A system architecture that would be able to fully utilize a decentralized network is a decentralized system architecture. A decentralized system is a system of nodes that is:

- robust – nodes are not affected when one area of the system fails to operate,
- scalable – the system allows for the addition of nodes without hindering the performance of the system,
- adaptive – the system adapts to the overall system settings [6].

In decentralized systems, each node possesses the same ability to process the system’s tasks, and maintains a copy of the system’s information. As shown in Figure 1.12, through these abilities, systems no longer experience the bottleneck of having nodes access a centralized area since each node is capable of acting as both a server and a client.
Figure 1.12 Decentralized information and decentralized processing.

1.5 Distributed Systems

A distributed system builds on the concept of a decentralized system. Like a decentralized system, a distributed system does not require a centralized area for information storage and processing. However, the difference between a distributed system and a decentralized system is that information is distributed throughout the network such that each node contains a unique subset of the total network information; the sum of all the information contained in the nodes will be the complete set of network information, as illustrated in Figure 1.13.
The figure also shows that distributed systems perform distributed computing by delegating the network tasks throughout the network. Although different nodes are responsible for different tasks, if the jobs are subtasks of a bigger job, then the nodes will have to work with one another to coordinate the completion of the tasks.

Any platform with a network of resources can perform distributed computing. Whether the resources are centralized in one location of the network or decentralized throughout the network is immaterial [7]. By using distributed systems to perform distributed computing, network nodes are no longer restricted to available computing resources at
The ability of a distributed system to spread information and processing capability throughout the network is the reason that a distributed system, unlike a centralized system, is fully able to utilize a distributed network infrastructure. Referring to the issues discussed in Section 1.3, a distributed system is better in that:

- Since information is processed throughout the network, there is no individual node that is burdened with the task of having to perform all the processing. When the network is scaled upwards in size, additional nodes actually add to the processing capability possessed by a network, as opposed to burdening the network even further.

- Since processing is performed throughout the network, the system is able to tolerate faultiness within the system and within the network. If the system collapses on one node, the processing can be performed on another node. If the connection is bad in an area of the network infrastructure, the information can be routed to the destination through different nodes.

### 1.5.1 Present Multi-Agent Frameworks

Presently, distributed systems are built upon the foundation of a multi-agent architecture. Called multi-agent systems, such systems are a collection of agents, which perform a set
of tasks or satisfy a set of goals [9]. In collaborating with one another to perform the tasks and to satisfy the goals, the agents communicate with each other via a specified communications protocol that is designed and incorporated into the architecture. The types of agents that exist within the distributed system will vary as it depends on the type of multi-agent architecture that is being used. The following subsections discuss the different available types of multi-agent frameworks:

- Cognitive Agent Architecture (Cougaar),

- FIPA Abstract Architecture,

- Jack-in-the-Net Architecture (Ja-Net),

- Java Agent Development Framework (JADE),

- JXTA.

1.5.1.1 Cognitive Agent Architecture (Cougaar)

The product of an eight-year effort to explore the potential of distributed multi-agent systems for military logistics, Cougaar is a Java-based architecture that is designed for the development of large-scale distributed agent-based applications [12] [13] [14].
The Cougaar agent is the main element in the Cougaar architecture and consists of two primary core components: a blackboard and plug-ins. A blackboard is an area in memory that is shared by subsystems within the agent. The different subsystems can read and write into this area. A plug-in is a software component that provides behaviour and business logic to the agent. This is beneficial because through the plug-ins, an infrastructure based on Cougaar technology can be incrementally enhanced as developers are able to keep developing and adding new plug-ins to the agent. Figure 1.14 represents an agent on a node.

In organizing Cougaar agents, a collection of unique agents that collaborate to solve a problem or a class of problems is called a Cougaar society. In a Cougaar society, several agents may execute on a node, which in Cougaar is a single Java Virtual Machine that can contain multiple agents. Agents on this node will share the same resources such as the processor, the memory pool, the disk, and the communication channels. It is important to note that allocation of agents to nodes is not necessarily domain-related, but rather could be related to the available resources. This advantageous because agents that require a large allocation of resources will not be denied those requirements.

Inter-agent communication is handled by the Cougaar Message Transport Service (CMTS), which is an adaptive Java Virtual Machine level service that is created when a node starts an agent. Although the CMTS is quite different from FIPA standards, it is component-based so that adaptive features can be selected at runtime. Standard protocols supported by Cougaar include RMI, CORBA, HTTP, and UDP. When an agent wishes
to communicate with another agent, the agents need only specify the details of the destination agent, and the system will deliver the message to the desired target [15]. The details may range from an agent’s name, which will be unique to each agent, to an agent’s attributes.

Figure 1.14 Example of Cougaar node architecture.

1.5.1.2 FIPA Abstract Architecture

The Foundation for Intelligent Physical Agents (FIPA), an organization under the IEEE Computer Society, developed an abstract architecture to set a standard that allows agents using different communications schemes to communicate with each other. The different
communications schemes may be different in messaging transports, in Agent
Communication Languages, or in content languages [16]. The standard included models
that would:

- allow agents to list their services so that other agents may be made aware of
  available services,

- allow for messages to be transferred from one agent to another using different
  messaging transports,

- allow for the supporting of various types of ACL representations,

- allow for the supporting of various forms of content language,

- allow for the supporting of multiple directory services representations.

With a strong focus on managing multiple message transport schemes, managing
message encoding schemes, and locating agents and services via directory listings, the
FIPA Abstract Architecture’s strength is that it provides an architectural overview of
agents, services, and directory listings. Hence, the FIPA Abstract Architecture focuses
on the functionality that such a system should have, as opposed to its structure. For
instance, upon starting a new agent, the agent should register its unique name with a
listing of agents, as well as register its own set of available services with a listing of
services. By performing these two simple procedures, other agents in the system not only
are able to view what other agents are available for communicating with, but also are able to view what services those agents provide.

In setting the standard for agent messaging, the FIPA Abstract Architecture provides the message structure to which systems should conform. The structure of the message is written in an agent-communication-language such as FIPA ACL, whereas the content of the message is written in a content-language such as KIF or SL. The message will also contain the sender and receiver information in the form of the unique names registered by the agents with the listing of agents. When a message has been created and sent, it is first encoded into a payload, and then included into a transport message. With this structure in place, the FIPA Abstract Architecture has provided a standard for the agents to communicate with one another - the content itself may only be understood by the collaborating agents. However, the message can be received and interpreted as an agent message by every agent because of the structure that was used to package the message.

1.5.1.3 Jack-in-the-Net Architecture (Ja-Net)

The Jack-in-the-Net Architecture is an approach that is used to design adaptive network applications in large-scale networks [17] [18]. The environment created by Ja-Net, known as a Ja-Net service environment, is a virtual network consisting of Ja-Net nodes. A Ja-Net node is a network-aware computing device, such as a PC or PDA, which has Ja-Net in its system. In addition, a Ja-Net node also provides a runtime environment and
support facilities for cyber-entities (CE), which are software with simple behaviours such as: migration, replication, reproduction, relationship establishment, and death.

From the perspective of a multi-agent system, a cyber-entity would be the agent in such a system. Each cyber-entity has a set of services, and depending on the service, will perform a sequence of behaviours based on a set of behavioural rules. Figure 1.15 illustrates the structure of the cyber-entity incorporated with the other components to form a Ja-Net framework.
Because the cyber-entities throughout the Ja-Net service environment collaborate to provide an application, there is a need for cyber-entities to communicate with one another. When a message is sent from one cyber-entity to another, the recipient of the message interprets the message and invokes the service action that corresponds to the message content. This action may in turn send a message to the initial cyber-entity or even another cyber-entity, developing a conversation amongst the CEs. The communication that is involved in the sending and receiving of the messages is based on the FIPA ACL, with extensions specific to Ja-Net as a communication language of cyber-entities. Included in this foundation is a set of keywords which are designated to denote key information in the messages. Examples of the keywords are: :sender, receiver, in-reply-to, sequence-id, and content.

1.5.1.4 Java Agent Development Framework (JADE)

JADE is a middleware that is used for the development of distributed multi-agent applications [10]. Developed by Telecom Italia Labs, JADE is based on the following advantageous driving principles:

- Interoperability – Agents created in JADE can collaborate with other agents as long as the other agents are using the FIPA standard specifications for communications.

- Uniformity and Portability – JADE provides an extensive set of APIs that are independent of the underlying network infrastructure and Java version.
• Simplicity and Usability – JADE provides an extensive set of APIs that are easy to use and hence, hide the complexity of the middleware.

• Required Learning – JADE has many features, but developers only need to learn about these features when the features are required for development. Furthermore, if a developer does not need the features, not understanding the features will not add additional computational overhead to the system [10].

As a result of the driving principles, JADE consists of two core components - a platform that enables developers to create FIPA-compliant agent-based systems, and a Java package that is used to develop software agents for inter and intra-platform communication between agents. Since JADE’s communications are based on FIPA standards, there are three agents that are incorporated in the platform [11]. The three agents are:

• Agent Management System (AMS) – an agent responsible for managing the platform’s operations such as creation, deletion, and migration of an agent.

• Directory Facilitator (DF) – an agent responsible for providing the “yellow pages” services to other agents. The “yellow pages” provides a description of the agents and the services the agents offer.
• Agent Communication Channel (ACC) – an agent responsible for using
  information provided by the AMS, and routing messages between agents,
  regardless of whether the routing is performed inter or intra-platform.

In the JADE platform, every instance of JADE is called a container. Although there can
be multiple containers as a result of multiple instances of JADE, only one container in the
platform may contain both the DF and AMS; this container is created upon the first
launching of JADE. Furthermore, since there is only one DF and one AMS, agents
residing on other platforms must rely on constant and reliable communication with that
container to be able to engage in a complete JADE runtime-environment.

Communications in JADE, summarized in Figure 1.16 and in Figure 1.17, is
distinguished as follows:

• agents residing in the same container and in the same platform communicate with
each other via events

• agents residing in different containers but in the same platform communicate with
each other via Java’s Remote Method Invocation (RMI)

• agents residing in different platforms communicate with each other via Internal
  Message Transport Protocols (IMTP) such as HTTP and WAP.
Figure 1.16 Intra-platform communication in JADE.
Figure 1.17 Inter-platform communication in JADE.
1.5.1.5 JXTA

JXTA is a framework that provides a set of protocols to support the development of
distributed applications. Currently, the latest version of JXTA is Project JXTA 2.0,
which builds upon five virtual network abstractions introduced in Project JXTA 1.0. The
five abstractions are:

- logical peer addressing – a model that spans the entire JXTA network,
- peer groups – these allow network participants, also known as peers, to
dynamically organize into protected virtual domains,
- advertisements – these are used to publish peer resources,
- resolver – a universal binding mechanism to perform binding required in a
distributed system,
- pipes – virtual communication channels allow applications to communicate with
one another.

Using JXTA protocols, a virtual network is created, thus allowing the exchange of
information between peers on the network. The benefit of such a network is that there is
no need for the peers to understand or manage complex network topologies that could be
formed. In addition to creating virtual networks, the JXTA protocols are also used to
standardize the manner in which peers discover each other, self-organize into groups, discover peer resources, and communicate with one another [19].

The protocols are divided into two categories – core-specification protocols and standard-service protocols. Core-specification protocols must be implemented if the system is meant to be JXTA-compliant, whereas standard-service protocols are optional. This is beneficial because developers only need to implement a subset of the overall protocols. The protocols currently defined in JXTA are:

- Peer discovery protocol (standard-service protocol) – the protocol by which a peer makes known to other peers the services that it offers. It is also a protocol for a peer to discover what services other peers are offering.

- Peer resolver protocol (core-specification protocol) – the protocol by which a peer is able to send a resolver query to one or more peers, and receive a response to the query.

- Peer information protocol (standard-service protocol) – the protocol by which a peer may obtain status information about other peers.

- Rendezvous protocol (standard-service protocol) – the protocol by which peers can subscribe or unsubscribe to a propagation service.
- Pipe-binding protocol (standard-service protocol) – the protocol by which a peer can establish a virtual communication link with one or more peers.

- Endpoint routing protocol (core-specification protocol) – the protocol by which a peer can discover a route used to send a message to another peer(s) [20].

The architecture of the JXTA platform is modelled after the standard operation system, and consists of three layers – applications layer, core layer, and services layer, as shown in Figure 1.18. As indicated by the diagram, communication can only occur between two neighbouring layers; the application layer cannot communicate with the core layer. In order to access services offered by the core layer, the application layer must access the service layer to act as an intermediate communication object, meaning the service layer accesses services desired by the application layer and relays those services to the application layer.

In the core layer, the following entities are created: peer, peer group, end point, pipe, advertisement, and identifier [2]. Through the creation of these objects, the core layer provides the elements necessary to facilitate the building of a JXTA virtual network. Building upon the core layer, the JXTA service layer provides network services that include searching for resources on a peer, sharing information amongst peers, and performing peer authentication. Finally, to meet the needs of the end-user, the application layer builds on the services and resources made available by the service layer.
In building on those services and resources, the application layer provides the packaging of a complete JXTA solution.

![Layering in JXTA platform](image)

**Figure 1.18** Layering in JXTA platform.

In JXTA communications, there are three types of peers that facilitate the management of requests and communications. These three types are:

- Rendezvous peers – peers used to relay and search for advertisement requests,

- Router peers – peers used to establish a path to a destination peer,

- Gateway peers – peers used to relay messages between peers [21].
1.5.2 Shortcomings of Present Multi-Agent Frameworks

In Section 1.4.1, several multi-agent frameworks were discussed to convey an idea of the types of systems that could be built. This section continues the discussion on those frameworks by outlining the shortcomings that exist with those architectures: use of centralised containers, lack of message control, use of client/server technology, and memory/thread management.

1.5.2.1 Use of Centralized Containers

Frameworks similar to JADE employ the use of a centralized container, which holds necessary core components in order for the system to function. The downfall of using such an architecture is that the system cannot function without those core components. Furthermore, because other agents in the node depend heavily on a reliable connection to the main container, any failure of the container would result in a system failure.

1.5.2.2 Lack of Message Control

Several of the frameworks discussed involved using communications schemes that were controlled by the framework’s software package. For instance, architectures built using JXTA and JADE cannot handle the transporting of messages as the software handles it internally.
1.5.2.3 Use of Client / Server Technology

In several of the discussed frameworks, RMI was used to perform inter and intra-platform communication amongst agents. This is an example of the client-server technology that is being used in a system that is designed to be liberated from such concepts. There is thus a need to be able to perform the same functionality, but without the use of client-server technology.

1.5.2.4 Memory / Thread Management

Some of the discussed architectures involved designating each agent with its own thread, while some of the other discussed architectures involved designating a collection of threads to a collection of agents. Thread management is an issue that needs to be investigated because utilizing too many threads to perform processing places a heavy burden on the resources of the system, while utilizing too few threads to perform processing results in a slow system. As a result, there is a need to find an intermediate approach to ensure that neither extreme occur.

1.5.3 Present Distributed Systems

A distributed system provides a cohesive network where each node’s hardware, software, and human users are integrated to create an entity that is capable of collaboration, cooperation and coordination [22]. Currently, most distributed systems exist in the form of intelligent peer-to-peer systems that concentrate on optimizing the large-scale transfer
of digital information. This section will discuss two of these systems – BitTorrent and Gnutella.

1.5.3.1 BitTorrent (BT)

BitTorrent is a peer-to-peer application that strives to replicate efficiently contents throughout a large set of peers by measuring the bandwidth of the peers [23]. In using BitTorrent, peers have two states – the leecher state, which represents the downloading of a content, and the seed state, which represents when a peer has a complete set of content.

The foundation of BitTorrent is based on the concept of a torrent – a term that defines a session of transfer of a single content to a given set of peers. A torrent is considered to be active as long as there is one seed in the session. This means that as long as there is a source that still has the desired content, the session will remain active. In BitTorrent, a file is typically divided into pieces of 256 kB, where each piece is subsequently divided into blocks of 16 kB. Note that it is possible to change the sizes of the divisions.

When a peer wants to obtain a file, the peer will download a .torrent file, usually obtained from a web server. This torrent-file provides information about a tracker. The tracker is a centralized component in BT that keeps track of peers involved in the transfer session, but does not play a role in the transferring of the file. Upon joining a transfer session, from the tracker, the peer connects to a list of peers involved with the transferring of the desired file and begins downloading the file. Since the file has been split into multiple pieces, a peer will not initially have all the pieces of the file, and so when using BT, each
peer is aware of which peer in the torrent has which piece. Collectively, the group of peers has the complete set of pieces that put together a file. When a peer does have a complete set of the pieces, the peer becomes a seed. Figure 1.19 shows an example of peers using BitTorrent.

The intelligence that exists within BitTorrent determines how the pieces within the torrent are transferred to the peers. This intelligence exists in the form of algorithms; two core algorithms that BT uses are the choke algorithm and the rarest first algorithm. The former guarantees that there is a certain level of upload and download reciprocation. This alleviates the problem of having any peers existing in the torrent who only download and never upload any pieces. In distinguishing up loaders and passive up loaders, the choke algorithm penalizes those peers not actively involved in uploading the pieces.

The second algorithm used in BitTorrent, the rarest first algorithm, is much simpler in comparison to the choke algorithm. This algorithm involves the local peer maintaining the number of copies of each content piece in a peer set. By doing this, BT defines a rarest pieces set. When a local peer has downloaded less than four pieces, the peer randomly decides the next piece to download. However, once the local peer has downloaded more than four pieces, the local peer randomly downloads the next piece from the rarest pieces set.
Figure 1.19 Example of an executing BitTorrent.
1.5.3.2 Gnutella

Gnutella is a pure peer-to-peer file-sharing application that allows Gnutella peers to connect with their neighbours using point-to-point connections [24] [25]. This decentralized model is a message-based system where peers forward queries throughout the entire Gnutella network. This means that every time a node sends out a query, every node in the network will receive it. In addition, because there are many flavours of Gnutella software, there are many possible ways of interpreting the query. For instance, some Gnutella software look into the file content being shared, whereas some Gnutella software look at the filename that is being shared [26]. As a result, a query may be interpreted in many different ways, which in turn could lead to different possible events that could be triggered in response to the query. Figure 1.20 and Figure 1.21 summarize query requests and query responses respectively.
Figure 1.20 Query requests in Gnutella.
Figure 1.21 Query responses in Gnutella.
The intelligence within the Gnutella framework exists in the collection of Gnutella software that each node possesses. Because each node could have different Gnutella software analyzing the messages, and because each node receives the query request and processes the query request using its own Gnutella software, the node originally sending the query request could have a variation of query responses. This is significant because it indicates that the intelligence used to analyze the query request is spread throughout the Gnutella network and not focused in one area.

1.5.4 Shortcomings of Present Distributed Systems

Although the distributed intelligence systems (BitTorrent and Gnutella) discussed in Section 1.5.1 reveal frameworks that are truly distributed and intelligent in nature, both systems have several shortcomings. Discussed in the following subsections, these shortcomings are related to a need for more cooperation and collaboration, a need for more intelligence, generated traffic flow, a lack of optimised caching, and a lack of optimised routing.

1.5.4.1 More Cooperation and Collaboration Needed

In BitTorrent, peers are only limited to viewing peers within the peer set of the torrent [23]. If there are peers existing in other peer sets that are capable of sending the file content at a higher efficiency, those peers will not be available for the exchange of information.
1.5.4.2 More Intelligence Needed

When transferring files from seeders to leechers, BitTorrent sends the same blocks to each seeder. This does not fully utilize the distributed capability of a distributed system. For instance, a better way to transfer the files would be to have the intelligence to send different blocks to different seeders - even-numbered blocks to one side of the network, and odd-numbered blocks to the other side of the network. Then, both sides can transfer blocks between themselves. Using this way of transfer would increase the efficiency of leechers becoming seeders.

1.5.4.3 Generated Network Traffic

Gnutella depends on flooding the network to relay query requests and responses throughout the Gnutella environment [27]. As a result, the load experienced by each node will vary depending on the size and configuration of the network topology. This load may require bandwidth beyond the limitations of a node, which would result in the fragmentation of the Gnutella network. Therefore, this presents a problem because fragmentation would trigger a consequence of eliminating a global search capability, which is Gnutella’s strong point.

1.5.4.4 Lack of Security

Gnutella does not provide any security layer to mask the identity of the user and the events of the user’s actions. Gnutella also lacks the ability to verify the contents of files.
These are significant issues because Gnutella would never be allowed to be incorporated into highly-sensitive systems such as financial systems and national defence systems.

1.5.4.5 Lack of Optimized Caching

A node that initiates a query request in the Gnutella network does not make the contents of its own cache known to the peers that are receiving the query request. As a result, there is no sense of what the original node has since its cache is composed of the results of the peers that are sending the results. Moreover, responding nodes do not know what other responding nodes are sending.

1.5.4.6 Lack of Optimized Routing

Gnutella lacks optimized routing that would allow peer-to-peer applications the ability to dynamically correct load imbalances as they occur. Imbalances could potentially affect the result of the whole Gnutella network since one part of the network may be heavily exhausted while another part may be processing nothing at the time.

1.6 Scope Of Thesis

1.6.1 Objective

The shortcomings outlined in Section 1.5.2 and Section 1.5.4 reveal the need for a new type of distributed system. That system, the Holonic Technology Platform (HTP), is the
objective of this thesis. The HTP is a distributed intelligence framework based on a hybrid model of multi-agent systems, utilizing holons, which will be explained in the coming chapters, and agents to perform and satisfy both local and global tasks and goals. Furthermore, the HTP is developed in Java because Java allows for P2P agent development. In addition, Java is platform independent, multithreaded, and easily packetized, meaning that Java code exists in the form of transferable byte code. In the next two chapters, the design of the HTP and the implementation of the HTP will be discussed. Furthermore, these next two chapters will also show how the HTP overcomes those previously-discussed shortcomings. Figure 1.22 provides a grading scheme comparing the infrastructures and systems discussed in this chapter with the Holonic Technology Platform. The comparison grades the distributed nature and intelligence of the infrastructures and systems.
1.6.2 Outline

Chapter 1 suggests potential problems with centralized infrastructures in networks and systems and suggests how these problems can be alleviated with the use of decentralized
infrastructures. Next, the chapter provides an overview of present-day distributed environments and present-day distributed intelligence systems. This overview includes a discussion of the shortcomings that exist in those distributed environments and distributed intelligence systems. Finally, the chapter concludes with a discussion on a potential solution to those shortcomings.

Chapter 2 discusses the design of the Holonic Technology Platform. There are two phases to this chapter – phase one being a discussion of the HTP’s system goals and objectives, and phase two being a discussion of the HTP’s infrastructure.

Chapter 3 discusses the functionality, interface and intelligence issues that were raised during the development of the Holonic Technology Platform. In deliberating those issues, needs for certain functionality are revealed. Those needs are also discussed in this chapter to exhibit how the HTP addressed those needs.

Chapter 4 demonstrates the step-by-step building of an environment created by the Holonic Technology Platform. Screen shots of console outputs and screen shots of debugger steps are used to clearly illustrate the incremental progression in creating the environment.

Chapter 5 summarizes the findings that were revealed during the development of the Holonic Technology Platform. Furthermore, this chapter also discusses future improvements that could be made to the HTP.
2 ARCHITECTURE AND DESIGN

This chapter describes the design and the architecture of the Holonic Technology Platform. Firstly, the system objectives and the implementation objectives will be discussed. System objectives describe the goals that are necessary to ensure that the HTP functions in a certain manner, whereas the implementation objectives specify how these goals will be realized. Secondly, the architecture of the platform will be explained with reference to the functionality of each of the different system packages. Finally, this chapter will conclude with a discussion of how the system design and system architecture will be integrated to provide the system with intelligence.

2.1 System Objectives

The Holonic Technology Platform has a set of system objectives that will govern the environment’s behaviour. These objectives will serve as the backbone from which the system’s architecture will be derived, and will serve as a paradigm for the development of the system’s functionality. The set of system objectives are:

1. To create an environment where each individual node on a network is able to work with other network nodes in accomplishing a global task. In such an environment, the individual must also accomplish its own given task.
2. To allow each individual node on a network to have the capability to learn new responsibilities and new sets of functionality, and to have the capability to adapt to a changing network environment.

3. To create a system architecture where other system developers are able to expand the functionality of the system without having to make changes to the core of the environment.

### 2.1.1 Accomplishment of a Global Task

Every task has an associated required effort and an associated required intelligence that must be fulfilled in order for the task to be completed. The Holonic Technology Platform is designed with the intention of creating an environment that distributes the effort and intelligence throughout a network. Presently, multi-agent systems incorporate nodes that are fixated with accomplishing their individually-assigned set of tasks. Such systems are only feasible when the nodes are not working together to achieve a global task. Figure 2.1 is an example of a scenario where nodes are working on tasks that are not related, whereas Figure 2.2 is an example of a scenario where nodes are working on tasks that contribute towards a global network objective.
Figure 2.1 Scenario of nodes not working towards a global objective.

Figure 2.2 Scenario of nodes working towards a global objective.
Figure 2.1 shows how present-day multi-agent systems would be sufficient in accomplishing the different tasks on a network. The nodes do not have to strive to achieve a common goal, and hence do not need to cooperate with one another. In Figure 2.2, however, there is a need for nodes to have the ability to cooperate with each other. If the nodes do not collaborate with the other nodes, then there is no way to gauge how successful the network is in attaining the global goal.

Currently, an agent is defined as a physical or virtual entity whose behaviour leads to satisfying its objectives, by taking into account the resources and skills available to it and depending on its perception, and by taking into account its representation and the communications it receives [29]. This means that the agent is solely focused on satisfying its local objectives. More importantly, this means that the agent is incapable of allowing nodes to cooperate with other nodes in attempting to achieve a global network goal. Instead of being modelled after an agent, a node should be modelled after a holon. A holon is an agent that has the capability to work with other holons in trying to achieve a global network goal [30]. A system that bestows nodes with holonic capability would be able to provide an environment where both the local tasks and the global tasks can be realized.

In summary, a holon can be an agent but an agent cannot be a holon. This is because an agent does not have the ability to communicate with other agents. Moreover, in attempting to achieve a global task, a holon may sacrifice the progress of completing its local tasks, perhaps by halting the procedure or by slowing down the process, if it means
that the global task will be achieved more efficiently. An agent, on the other hand, does not care about the performance of the overall goal, as it only cares about the completion of its own local tasks.

2.1.2 Adapting to a Changing Environment

As the Holonic Technology Platform increases in functionality and in complexity, there may be system components that have to be updated out of necessity, and to a certain extent, be added to the system. The key, however, is to perform these adjustments while the system is still performing. As a result, there is a need for a holon to be composed of agents. Within the holon, there are many different tasks that have to be managed. Agent technology would be suitable for this scenario because the agents would be designated to accomplish a task that is unique to others, and would be loosely-connected with one another. Figure 2.3 best demonstrates how different tasks within a holon would be coordinated using agents.

![Figure 2.3 Coordination of tasks within a holon.](image)
The benefit of this system architecture is that if an agent component has to be changed, the other agents are not affected. For instance, if Agent 4 has to be updated with new functionality, the other three agents are still able to perform their own tasks as expected. Another example of this architecture’s usefulness is demonstrated when an agent that monitors network traffic has to be added to the system. The new agent is simply added to the list of agents and when it is performing, functions like the other agents in terms of handling its own task(s). Figure 2.4 shows the final agent-layout within the holon after this addition.

Figure 2.4 Agent layout after addition of an agent.
2.1.3 Designing for Expansion

Tools must be developed concurrently with the development of Holonic Technology Platform. An environment that does not have development tools will not be able to expand rapidly and efficiently. These tools should allow other system developers the flexibility of adding system functionality without affecting the core of the system. Only in this manner, can applications be developed properly such that they are able to fully utilize the abilities of the environment.

2.2 Implementation Objectives

Implementation objectives are defined to serve as a set of guidelines to which the architecture and functionality of the Holonic Technology Platform must conform. The following is a list of implementation objectives that will be discussed:

- Functionality within the HTP must be kept specific to either the Holonic Logistics System or the Holonic Strategic System.

- Communications within the HTP must be divided into internal and external communications.

- A standard message structure must be used for all communications protocols.

- Architectures of agents and holons must be standardized.
• Architecture of application using HTP must be standardized.

• System must be able to manage multiple virtual networks.

• System must implement distributed methodology in distributing effort, intelligence, resources, and management capability.

• Holons must be able to dynamically acquire intelligence.

2.2.1 Holonic Logistics System versus Holonic Strategic System

To allow for faster development, the Holonic Technology Platform has been partitioned into two separate layers – the Holonic Logistics System (HLS) and the Holonic Strategic System (HSS). The Holonic Logistics System is responsible for the overall management of the platform, whereas the Holonic Strategic System is responsible for making the appropriate decisions given the current state of the network environment. Furthermore, this partitioning clearly defines the roles and responsibilities of the two systems, so that there is no overlap in functionality. To communicate with other nodes, the platform will also have a Holonic Communications System (HCS). Presently, because the Holonic Communications System has not been completed, a platform that simulates a wireless distributed environment is being used to allow for the continuous development of the Holonic Technology Platform.
Figure 2.5 shows the interaction between the three systems within the platform, where the Holonic Strategic System only communicates with the Holonic Logistics System, and where the Holonic Logistics System only communicates with the Holonic Communications System. Furthermore, because the strategic duties has been separated from the logistics duties, as illustrated in Figure 2.6, the Holonic Strategic System will be viewed as an application by the Holonic Logistics System. From this point onwards, we will be addressing the Holonic Strategic System as a software application using the Holonic Logistics System. It should also be noted that this document focuses on the development of the Holonic Logistics System.

Figure 2.5 Layers in the Holonic Technology Platform.
2.2.2 Internal Communications versus External Communications

There are two types of messages that are passed throughout the Holonic Technology Platform. Internal messages are defined as messages that travel within the HLS. External messages are defined as messages that travel from one holonic layer to another holonic layer, and as messages that travel between holons. Table 2.1 summarizes the difference between the two types of communications.
Table 2.1 Internal Communications vs. External Communications

<table>
<thead>
<tr>
<th>Internal Communications</th>
<th>External Communications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication between agents</td>
<td>Communication between holonic subsystems</td>
</tr>
<tr>
<td>within holonic subsystems</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Communication between holons</td>
</tr>
</tbody>
</table>

Internal communications within a holonic subsystem will be implemented using events. When information has to be transferred or when a signal has to sent, an event will be triggered. This triggered event will then be handled by the agent switchboard in the holonic subsystem, which is responsible for relaying data between agents, allowing them to communicate with each other while remaining loosely connected with one another. Figure 2.7 shows the agent switchboard within the holonic subsystem.

![Agent Switchboard](image)

Figure 2.7 Agent switchboard used for internal communications [31].

*The Agent Switchboard will listen for all events and based on the desired functionality, select the appropriate agent to handle the event.*
To facilitate the bidirectional external communication between the Holonic Logistics System and the other holonic layers, there will be two sets of input and output ports dedicated to communicating with the Holonic Strategic System and with the Holonic Communications System. The mechanism used to transfer the messages between the systems will be UDP. In addition, internal within the holonic subsystems, there will also be a mechanism that will categorize incoming external messages as either administrative messages or user messages. Figure 2.8 is a representation of the communications interface.

![Figure 2.8 Interface for external communications.](image-url)
2.2.3 Standard Message Structure

External messages sent throughout the Holonic Technology Platform will adhere to the message structure in Figure 2.9. Details pertaining to the processing of the message based on this structure will be discussed later in this chapter.

![Figure 2.9 Message structure of external messages.](Image)

2.2.4 Agent Architecture versus Holon Architecture

Figure 2.10 defines the relationship between a holon and an agent in the Holonic Technology Platform. As mentioned earlier in this chapter, the functionality of an agent within the HTP is very specialized since each agent represents a subsystem within the holon. However, even though each agent may be different, there is still a need to keep the core of an agent uniform. Keeping an agent’s core generic allows for easier agent development since every agent must have certain common features. The UML class diagram in Figure 2.11 indicates the architecture of the generic core. Although the class diagram represents a relatively simple core, the essence of having a set of commonalities
within the agents is very much present. As the Holonic Technology Platform advances in complexity and in functionality, this generic core will also do the same.

Figure 2.10 Relationship between holon and agents.

Figure 2.11 Generic architecture of an agent’s core.
Because a holon’s architecture is a collection of agents, a core of agents must be defined. Each holon will be different depending on the application functionality that the holon possesses. The core of agents guarantee that the holon, despite the disparity that could exist in application functionality, will still be able to communicate and cooperate with one another to perform a core set of tasks such as logging onto the environment and creating links with other holons. The 12 agents that exist in the core of a holon are shown in the following UML class diagram Figure 2.12. In the figure, because this thesis focuses on the Holonic Logistics System, references made to a holon refers to the HLS. The functionality of each core agent will be discussed in the upcoming sections.

Figure 2.12 Agent packages in the Holonic Logistics System.
2.2.5 Holonic Application Architecture

Holonic applications are applications that utilize the environment provided by the Holonic Technology Platform. As is the case with agents and holons within the HTP, holonic applications will have a core that is generic to all holonic applications. The relationship between holonic applications and holons is shown in Figure 2.13.

![Diagram showing the relationship between holonic applications and holons.](image)

Figure 2.13 Relationship between holonic applications and holons.

As Figure 2.13 indicates, a holon may have several holonic applications that are connected to it. However, although the holons exist on the same environment, it does not guarantee that the same holonic applications will exist on every holon. The benefit of having a core that is common to all holonic applications is that regardless of the
functionality of the application, the holon will not have to understand the communication protocols that could be unique to all applications. As a result, the purpose of the core of a holonic application is to allow for a standard communication mechanism between the application and the holon to which it is connected.

Because holonic applications do in fact cooperate and coordinate tasks amongst one another in the holonic environment, the holonic applications can be conceptually considered a holon as well. Because of these two points, the architecture of a holonic application will be modelled after the architecture of the HTP’s holon, with the exception that the former will be much simpler. Figure 2.14 describes the architecture of a holonic application.

Figure 2.14 Agent packages in a holonic application.
2.2.6 Management of Multiple Virtual Networks

The Holonic Technology Platform supports the use of virtual networks to better organize the holons within the holonic environment. A virtual network (VNET) is defined as an interconnected group of networks (an internet) that appears as one large network to the user [32]. The HTP extends this definition by adding that connections between holons within a virtual network are logical connections. Figure 2.15 shows the relationship between a physical connection and a logical connection.

![Physical Connection](image1.png) ![Logical Connection](image2.png)

Figure 2.15 Relationship between a physical connection and a logical connection.
As indicated by the diagram, a logical connection requires a physical connection. However, physical connection between two holons does not necessarily guarantee a logical connection between them. As a result of this difference in connectivity, the HTP offers users the possibility of configuring different network topologies based on a pool of available holons. Figure 2.16 shows how a user may utilize this capability.

![Figure 2.16 Examples of virtual layers.](image)

The above figure shows that holons are able to belong to different virtual networks. This point may seem redundant to focus upon, however, this leads to the HTP's ability to support large scale networks. Figure 2.17 demonstrates how the Holonic Technology Platform can be expanded to span large global networks.
In addition to using virtual networks to organize the nodes into clusters, the HTP also utilizes the virtual networks to simulate a wireless environment. The different virtual networks are capable of representing the different dynamic wireless configurations.

2.2.7 Distribution of Network Resources and Network Intelligence

The Holonic Technology Platform implements the concept of distributing all network tasks, network wealth and network intelligence throughout the entire holonic environment. As mentioned in Section 1.5, by distributing the resources and knowledge of a network amongst the connected holons, this not only imposes less stress on the physical resources of the individual holons, but also offers benefits such as protection against network failure and security against malicious system attacks. The distributed approach that the HTP uses to manage its holonic environment will be explained in the upcoming sections. However, some holonic functionality within the HTP will be
implemented with a centralized concept because the centralized approach is simpler and more efficient.

2.2.8 Dynamic Acquisition of Intelligence

The Holonic Technology Platform has been designed to have the ability to cooperate with other holons, and to have the ability to coordinate the completion of a set of global tasks. Furthermore, the Holonic Technology Platform has also been designed to allow expansion of its local intelligence. These implementation objectives offer a user of the holonic environment an array of possibilities in terms of how the user wishes to coordinate applications within the holonic environment. An important point to discuss is how a holon will learn to adapt to a changing holonic environment while connected to the platform and performing tasks. There is a need to allow a holon to be able to dynamically acquire intelligence, as demonstrated in Figure 2.18.
Figure 2.18 Need for ability to acquire intelligence.
### 2.3 Holonic Agents in the Holonic Technology Platform

Table 2.2 Agents in the Holonic Technology Platform [33]

<table>
<thead>
<tr>
<th>Agent Name</th>
<th>Agent Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administration Agent</td>
<td>Monitors and controls system administration functionality such as system shutdown, system update and system initialization.</td>
</tr>
<tr>
<td>Algorithms Agent</td>
<td>Initiates algorithms and manages settings of algorithms within the Holonic Logistic System</td>
</tr>
<tr>
<td>Applications Manager Agent</td>
<td>Monitors and controls all functionality related to an application registered with the system.</td>
</tr>
<tr>
<td>Console Agent</td>
<td>Displays system outputs to the user and accepts system inputs from the user.</td>
</tr>
<tr>
<td>Database Agent</td>
<td>Backs up the database of each agent before system shutdown and restores database of each agent during system initialization.</td>
</tr>
<tr>
<td>Development Agent</td>
<td>Creates new agents and holonic applications</td>
</tr>
<tr>
<td>Message Analyzer Agent</td>
<td>Analyzes all incoming and outgoing messages for internal and external purposes.</td>
</tr>
<tr>
<td>Messenger Agent</td>
<td>Sorts all incoming and outgoing messages for internal and external purposes.</td>
</tr>
<tr>
<td>Software Transfer Agent</td>
<td>Checks to ensure that each node is updated with the latest software version of the Holonic Logistic System</td>
</tr>
<tr>
<td>System Agent</td>
<td>Monitors operating system attributes such as number of threads, number of running applications, etc.</td>
</tr>
<tr>
<td>Version Tracking Agent</td>
<td>Keeps track of software versions within the Holonic Logistic System</td>
</tr>
<tr>
<td>Virtual Networks Manager Agent</td>
<td>Monitors and controls all functionality related to a virtual network.</td>
</tr>
</tbody>
</table>
Each agent subsection will employ the use of UML use-case diagrams, UML sequence diagrams, and UML activity diagrams to show the detailed functionality of each agent. These agent subsections, however, will not discuss the implementation of the functionality which instead will be discussed in the next chapter. There are three categories of functionality that will be shown:

- **holonic application functionality** – functionality performed by the holonic application

- **HLS functionality** – functionality performed by the Holonic Logistic System

- **user functionality** – functionality performed by the user of the HLS

There are functions that will exist in more than one category because the holonic application can possibly have the option of triggering mechanisms that exist within the holonic logistics system.

### 2.3.1 Administration Agent

The Administration Agent’s main responsibilities are to prepare the system for shut-down and to prepare the system for initialization. When shutting down the system, the Administration Agent first has to ensure that all vital system information is saved into the HLS’ database. To do this, the Administration Agent has to coordinate the procedure with the Database Agent. As mentioned previously, because holonic agents coordinate
with each other via the use of events, an event is sent to the Database Agent to begin the process of backing up the data. When the system backup has been completed, an event will be triggered to notify the Administration Agent to continue with the shutdown procedure. Figure 2.19 is a UML sequence-diagram that illustrates the sequence of events that occur during the shutting down of the system.

![Figure 2.19 Logic flow of system shutdown.](image)

The initialization of the system is slightly more complicated in that the Administration Agent has two scenarios to consider – one where the HLS is being initialized for the first time, and one where the HLS is being initiated after a system reboot. If the system is running for the first time, the Administration Agent performs a series of tests to ensure that the system is functioning in the expected manner. Upon completion of the tests, the user will be able to configure the holon as desired, and holonic applications will be able
to connect to the holon. If the Administration Agent is initializing the system following a system reboot, it must trigger the alert the Database Agent to restore previously-saved vital system information. When the system has successfully restored, the Administration Agent will conduct routine startup tests, after which the users and holonic applications will be able to use the HLS. Figure 2.20 is a UML sequence diagram of the system initialization process.

![UML Sequence Diagram](image)

Figure 2.20 Logic flow of system initialization.

In addition to shutting down the HLS and initializing the HLS, the Administration Agent is also responsible for sending emergency signals to warn other holons of certain events within the holon, and is also responsible for synchronizing the HLS’ clock with atomic clocks throughout the world.
This latter mechanism is extremely important because it enables holons to be synchronized with other holons, ensuring that tasks that require coordination amongst holons will be more accurately and efficiently completed. Synchronization with the atomic clocks will become more frequent when there are many tasks requiring coordination within the holonic environment. Accordingly, when there are only a few of such tasks, the frequency of synchronization with the atomic clocks will be slower. Figure 2.21 is a UML use-case diagram that summarizes the set of functionality performed by the Administration Agent.

![Figure 2.21 Functionality of Administration Agent.](image-url)
2.3.2 Algorithms Agent

The functionality of this agent can be separated into two tasks – managing algorithms and managing algorithm variables. In managing algorithms, the Algorithms Agent is responsible for the integration of algorithms into the system and for the separation of algorithms from the system. By integrating the algorithms into the system, the Algorithms Agent also makes the algorithms available to users of the Holonic Logistics System by listing the algorithms in a library of algorithms. Finally, after an algorithm has been integrated, it can be executed by users through the Algorithms Agent. The implementation of these mechanisms will be discussed in the next chapter.

An algorithm variable is defined in the holonic environment as a variable that has one or more values of one or more of the following types: boolean, double, integer, long, and string. Different algorithms may have a number of associated algorithm variables. In managing algorithm variables, the duty of the Algorithms Agent is to associate the variables with the proper algorithms. Furthermore, the responsibilities of this agent also extend to modifying the value(s) of the variables and to listing the variables in the library of algorithm variables in the event that algorithms may share similar variables. Figure 2.22 is an UML use-case diagram summarizing the mechanisms performed by the Algorithms Agent. These mechanisms are considered to be user functionality because users trigger the mechanisms.
2.3.3 Applications Manager Agent

Figure 2.23 reveals the functionality within the Applications Manager Agent. As illustrated in the figure, the Applications Manager Agent is responsible for managing holonic applications that are registered with the Holonic Logistics System and for managing application services within the holonic application.

An application service is a functionality offered by a holonic application. For example, a holonic application that manages traffic within a virtual network may have the following application services: locating the high-traffic areas in the environment, monitoring the number of nodes in the environment, and monitoring the available collective resources in the environment.
In order for a holonic application to gain connection with a holonic environment, the application needs to register with the Holonic Logistics System. This only requires a single message from the holonic application to the Holonic Logistics System specifying to the HLS the name of the holonic application, the description of the holonic application, and the communication settings of the holonic application, such as the port numbers.

Figure 2.23 Functionality of Applications Manager Agent.
Modifying and deregistering holonic applications also only require a single message from
the holonic application to the Holonic Logistics System. The information specified to the
HLS is listed in Table 2.3.

<table>
<thead>
<tr>
<th>Task</th>
<th>Information Sent</th>
</tr>
</thead>
</table>
| Modifying holonic application     | • existing name of holonic application
|                                   | • new name of holonic application     |
|                                   | • existing port settings              |
|                                   | • new port settings                   |
| Deregistering holonic application | • holonic application name             |

Registering, deregistering, and modifying application services are performed almost in
the same manner as the holonic applications. The only difference is that holonic
applications have to first register with the HLS before the application is able to register
application services with the HLS. The sequence diagram in Figure 2.24 shows this.

Figure 2.24 Logic flow of registering a holonic application.
One of the mechanisms the Applications Manager Agent provides is searching the holonic environment for holons that have specific holonic applications, or even specific application services. This search mechanism allows applications to easily find services in a distributed network where there is no centralized database. Upon locating the application service(s) and the holonic application(s), the holons can then begin coordinating tasks with one another. Figure 2.25 demonstrates a scenario.

![Diagram](image-url)

Figure 2.25 Search mechanism in applications manager agent.
The Console Agent’s main responsibility is to coordinate information between the Holonic Logistics System and the user. As a result, every time information is relayed from the user to the HLS, or from a holonic agent to the user, an event is triggered to inform the Console Agent of the task. Figure 2.26 shows the detailed set of functionality performed by this agent. As indicated by the use-case diagram for this agent, the agent is also responsible for coordinating the flow of information between the GUI agent and itself and for logging user inputs. As an example, Figure 2.27 shows the sequence diagram of how an individual menu is shown for a specific holonic agent [33].

Figure 2.26 Functionality of Console Agent.
2.3.5 **Database Agent**

As mentioned previously, the duty of the Database Agent is to backup vital system information and to restore vital system information. UML sequence diagrams in Figure 2.28 and 2.29 show the process of backing up a database and the process of restoring a database [33]. As the sequence diagrams for this agent indicate, each holonic agent plays role in both the backing up of the Holonic Logistics System and the restoration of the Holonic Logistics System. In terms of the former, the individual holonic agents are responsible for relaying the most up-to-date information to the Database Agent. In terms of the latter, the individual holonic agents are responsible for configuring their agent settings when the Database Agent relays previously-saved information to them. The UML use-case diagram in Figure 2.30 summarizes the functionality of the Database Agent.
Figure 2.28 Logic flow of system-backup.
Fire an event to agent informing that Database Agent is now restoring system database.

Read database and initialise using saved data.

Figure 2.29 Logic flow of system-restoration.
2.3.6 Development Agent

Figure 2.31 illustrates the functionality performed by the Development Agent.
This agent is designed to allow system developers to have the capability to expand on the functionality of the Holonic Logistics System. Within the HLS, there are three levels in the organization of development – core development, proprietary development and holonic application development. Core development refers to any work developed to enhance the core of the Holonic Technology Platform. Proprietary development refers to any additional non-core functionality that is developed to enhance the Holonic Technology Platform. Holonic application development refers to the development of holonic applications.

By having the functionality outlined in the use-case diagram, developers have the freedom to add to the functionality of the HLS in an organized manner that is easy to keep track of. Furthermore, the set of functionality that creates new agents and new holonic applications provides a template of how the agent and holonic application should look like in terms of the software architecture. Also provided are the basic communications and holonic functionality required to allow easy integration between agents and holonic applications with the HLS. The onus is therefore on the developers to complete the rest of the agent’s functionality.

Agents are also able to join agent teams for better organization. For instance, if there is a group of agents that only coordinate with one another, the user may choose to categorize those agents as an agent team. An example is if there are five agents performing the following tasks: locate nodes with religious music, sort religious music, copy religious music, transfer religious music, and play religious music. Because the five agents all
perform tasks related to religious music, the agents may form a group called "Religious Music Agents".

### 2.3.7 Message Analyzer Agent and Messenger Agent

These two agents cooperate with one another to handle incoming and outgoing messages. The Message Analyzer Agent analyzes incoming messages to determine which holonic agent should service the message, and the Messenger Agent transfers the message to that corresponding holonic agent. In terms of outgoing messages, the Message Analyzer Agent tracks message details such as the destination and the purpose of the message.

To handle the large volume of messages flowing through a holon, a three-level hierarchy has been designed to allow for temporary caching of messages and to allow for better organization of message flow. Such a hierarchy exists for both incoming messages and for outgoing messages. Figure 2.32 shows the hierarchy for incoming messages.

---

Figure 2.32 Hierarchy for handling incoming external messages.
In order to simplify the explanation of the mechanism that manipulates the different levels of data structures, the situation being explained will be taken from the input message hierarchy’s mechanism. The highest level, Level 1, consists of one queue that stores all incoming messages. Whenever a message is placed into this message queue, an event is triggered to alert the Messenger Agent that there is a message waiting to be serviced. The Messenger Agent then retrieves a message from the front of the queue and places it into one of two queues in Level 2. Because the Holonic Logistics System is an intermediate layer between the Holonic Communications System Layer and the Holonic Applications Layer, there is a need to have a queue to separate messages into the appropriate layers. In this manner, there is at least some form of organization amongst the messages.

Similarly, when a message is placed into either the queue for the HCS or for the HSS, an event is triggered to alert the Messenger Agent that it needs to handle those new messages. When the event is handled, the Messenger Agent places the message into a queue in the lowest level, Level 3. The number of queues that exist within this level correspond to the sum of the number of holonic applications and the number of virtual networks that exist within the HLS; for every holonic application and for every virtual network, a new message queue is created for the specific object. When the message is placed into a Level 3 queue, once again, an event is triggered and the Messenger Agent will be notified that a message is waiting to be serviced in a Level 3. The difference, however, is that now the message will actually be serviced. This means that whatever mechanism the message was meant to triggered will be triggered and the HLS will handle
the mechanism via the appropriate holonic agents. The UML sequence diagram in Figure 2.33 summarizes the incoming mechanism.

![Figure 2.33 Logic flow of handling incoming external messages.](image-url)

The design of the hierarchy could have been simplified to just one queue where all messages are serviced immediately upon being received. However, if a system is required to shut down, messages within the queues may be lost. By having a three-level order, the queues are backed up one level at a time during system shut down in the order of Level 3, Level 2 and then Level 1. This allows Level 1 to keep receiving messages while the lower two levels are being backed up. Although this does not guarantee that all messages will be saved, it does decrease the number of lost received messages. There is also the benefit of decreasing the strain on a holon’s computational resources. By having several levels prior to servicing the message, the levels act as buffers that slow down the
processing of the messages. Though the delay is minimal, the small fraction of time
saved relieves a great burden on the HLS. The mechanism handling the output hierarchy
is similar to the mechanism handling the input hierarchy in that the messages are
propagating from one level to another. The difference, however, is that the messages are
propagating in the opposite direction from a lower level to a higher level. Figure 2.34
summarizes the output mechanism and Figure 2.35 summarizes the sets of functionality
performed by both holonic agents.
Figure 2.34 Logic flow of handling outgoing external messages.
2.3.8 Software Transfer Agent

The Software Transfer Agent's duty can be summarized into the single task of ensuring that software associated with the HLS is properly transferred from one holon to another. However, there is a set of different subtasks performed by this agent to ensure that the task is properly done, as outlined in Figure 2.36.
There are two types of software transfer that can occur – the transfer of a holonic application and the transfer of agents within the Holonic Logistics System. In the case of the former, all the files within the holonic application are transferred from one holon to another. In the case of the latter, via holons throughout the holonic environment, the agent is able to both update and add new software files for holonic core agents and for holonic proprietary agents. The protocol for transferring holonic applications is represented by a UML sequence diagram in Figure 2.37, whereas Figure 2.38 is a UML sequence diagram representing the protocol used in the transferring of holonic agent software.

Figure 2.37 Logic flow of transferring holonic applications.
Holon A

Request permission to begin software transfer mechanism

If Holon B is available to perform the software transfer mechanism, then permission is granted. If not, then permission is denied.

If Holon B granted permission to begin software transfer mechanism, Holon A will request for the list of source files and their corresponding version numbers. If not, then Holon A will continue to request for permission every 60 seconds until permission is granted or until Holon A decides to stop.

Send list of source files and corresponding version numbers

Compare the Holon B's list with own list, and decide which files should be copied from Holon B

Request for desired files

Send requested files

Check to ensure that newly-downloaded files are properly loaded into system. Update own list of source files with new updated version numbers. If new files were downloaded, add source file information into list.

Inform Holon B to initiate software transfer mechanism

Holon A and Holon B repeat steps between I and II. The difference in this second round of software transfer is that Holon B is updating its source files by comparing its own list with Holon A's list.

Inform Holon A that it has completed the software transfer mechanism

Figure 2.38 Logic flow of transferring holonic agents.
2.3.9 Systems Agent

The Systems Agent is responsible for acquiring the following system information:

- class-loading data
- compilation-system data
- garbage-collector data
- memory-system data
- memory-manager data
- memory-pool data
- operating-system data
- runtime-system data
- threading-system data

By being able to monitor the different resources on the holon, the user is able to see resource strain being placed on the system. In addition, within this agent exists a system manager that is responsible for each of the different types of information, allowing for better organization as the system is able to trace which manager is malfunctioning in the
event of a System Agent error. Figure 2.39 summarizes the use-cases of the System
Agent.

![Figure 2.39 Functionality of Systems Agent.](image)

**2.3.10 Version Tracking Agent**

The responsibilities of the Version Tracking Agent are simple, as illustrated by Figure
2.40. Every time the Holonic Logistics System is initialized, the Version Tracking Agent
is populated with the filenames of the source code that exist within the system and with
the version numbers of those files. This provides the HLS with a database of the
different software versions so that software transfers can be performed to update the older
versions. As mentioned previously, software versions are compared between two holons
during a software transfer. During the comparison, this agent cooperates with the
Software Transfer Agent by providing the data for the comparison mechanism. As the software transfer is being performed, the Version Tracking Agent is updated with the new version numbers of the new files. If new files are added to the system, they will be added to the database of the Version Tracking Agent.

![Diagram of Version Tracking Agent]

Figure 2.40 Functionality of Version Tracking Agent.

2.3.11 Virtual Networks Manager Agent

As discussed previously, virtual networks of various topologies can be created based on the physical connections that exist within the holonic environment. Because there may be traffic on more than one virtual network at any given time, there is a need to manage efficiently the virtual networks so that messages can be delivered to the proper destinations. As a result, an agent has been designed and developed to do the management, and Figure 2.41 shows the set of functionality performed by the agent.
The tasks that this agent is capable of executing are not difficult. However, if there is a necessity to organize holons into specific surroundings, as in the form of virtual networks, then there is the requirement that the virtual network environment first be set up. To do this, two things have to be done – the creation of resources for the virtual network on the holon and the creation of virtual link connections between the holons. In addition to creating the virtual environment, holonic applications connected to the holon and users of the HLS have the ability to modify the settings such as the link connections, to modify network settings such as delay intervals between message deliveries, and to remove the holon from virtual networks.
2.4 Distributed Intelligence in the HTP

As a standalone distributed system, the HTP depends on the holonic applications and the users of the platform to guide the functionality of the system. This means that the behaviour of the system is dependent upon the ability of the holonic applications and the ability of the holonic users to coordinate tasks. However, this standalone system offers many benefits in the sense that:

- the architecture of the system is based on the agent methodology, which stresses loose dependency between subsystems, with the added capability of allowing agents to cooperate with one another

- the system is able to simultaneously cater to multiple virtual networks that are created from a common pool of holons

- the system is created based on distributed methodology such that the system shares the network’s resources and knowledge amongst all the network nodes

To reduce the dependency on the holonic applications and on the holonic users, distributed intelligence is incorporated into the Holonic Technology Platform. This intelligence gives the HTP the ability to adapt to a potential changing holonic environment, and is integrated into the HLS through the implementation of intelligent distributed algorithms (to be discussed in subsections of the next chapter).
2.5 Summary

The Holonic Technology Platform is an environment that extends agent technology by allowing agent nodes within a network to cooperate and coordinate tasks with one another in an attempt to achieve a global goal. Functionally guided by system objectives and implementation objectives based on the distributed methodology, the platform’s architecture is a collection of agents that allows for increase in both system complexity and system expansion.
3 IMPLEMENTATION OF THE HLS

This chapter entails the implementation of the HLS. With the aid of UML diagrams, the diverse implementation issues encountered during the development of the Holonic Logistics System will be examined. The order of the issues to be discussed is as follows:

- functionality issues – implementation issues involving the behaviour of functions within the HLS
- interface issues – implementation issues involving the interfacing between the different layers within the HLS
- intelligence issues – implementation issues involving the integration of intelligence within the HLS.

These issues will be presented as a problem-solution set; the problems will first be defined, followed by employed solutions.

3.1 Functionality Issues

In implementing the functionalities of the Holonic Logistics System, several predicaments were encountered. The following subsections describe those predicaments.
3.1.1 Thread Management

The Holonic Logistics System utilizes multiple threads to accomplish its tasks. As Figure 3.1 shows, this means that several threads may be working on one assigned task.

![Thread Management Diagram]

Figure 3.1 Multiple threads working on a task.

The advantage of having several threads working on a task is that tasks are completed quicker. For instance, if a user is editing a spreadsheet document and calculations have to be redone to reflect those edits, the following tasks will occupy a thread: relaying input from user to spreadsheet system, recalculating spreadsheet outputs with new data, and saving new data [34].

If the spreadsheet system was to use a single thread to complete all these tasks, the completion time for the editing a spreadsheet would be much longer. The reason being is
that the system would have to switch between tasks to allow each task to finish progressively, and that the system would have to waste time waiting for the user to enter the data.

Despite a faster completion time, implementing a multithreaded system does raise concerns such as:

- consumption of memory,
- lack of processor time,
- corruption of data,
- consequences of killing threads.

The problem that arose during the implementation of the HLS was determining the best way to incorporate multi-threading into the system such that those concerns would be limited. The solution to this problem was to have two categories of threads within the Holonic Logistics System.

### 3.1.1.1 Use of Thread Resources

There are two categories of threads created by the HLS – system threads and event threads. System threads are created during system boot-up, designated to perform one
task and killed only when the system is shut down. Upon initialization of the HLS, three threads are created and designated for:

- communicating with the HSS
- communicating with the HCS
- displaying the user console.

Event threads, on the other hand, differ from system threads in that they are created when an event is called and killed when the event has been serviced.

The three system threads are a necessity because those tasks are essential to the performance of the HLS. Without communications threads, the HLS would have no form of communicating with other holons and other holonic applications. By having other system tasks executed by event threads, the system is optimized for efficiency in that both memory and processor time are only occupied when there is a task to be completed. When the task has been completed, the event thread handling the task is killed, resulting in the release of memory and process resources. On the following page, Figure 3.2 is an example of this efficiency.
<table>
<thead>
<tr>
<th>Time</th>
<th>Threads In System</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 seconds</td>
<td><img src="image" alt="System Thread" /> <img src="image" alt="Event Thread" /></td>
<td>System initialized</td>
</tr>
<tr>
<td>5 seconds</td>
<td><img src="image" alt="System Thread" /> <img src="image" alt="Event Thread" /></td>
<td>Event 1 triggered to complete a task.</td>
</tr>
<tr>
<td>10 seconds</td>
<td><img src="image" alt="System Thread" /> <img src="image" alt="Event Thread" /></td>
<td>Event 2 triggered to complete a task.</td>
</tr>
<tr>
<td>15 seconds</td>
<td><img src="image" alt="System Thread" /> <img src="image" alt="Event Thread" /></td>
<td>Event 3 triggered to complete a task.</td>
</tr>
<tr>
<td>20 seconds</td>
<td><img src="image" alt="System Thread" /> <img src="image" alt="Event Thread" /></td>
<td>Event 2 completed.</td>
</tr>
<tr>
<td>25 seconds</td>
<td><img src="image" alt="System Thread" /> <img src="image" alt="Event Thread" /></td>
<td>Event 3 completed.</td>
</tr>
<tr>
<td>30 seconds</td>
<td><img src="image" alt="System Thread" /> <img src="image" alt="Event Thread" /></td>
<td>Event 1 completed.</td>
</tr>
</tbody>
</table>

Figure 3.2 Use of threads in Holonic Logistics System.
3.1.1.2 Management of Data Corruption

In multithreaded systems, the synchronization of writing data to the system amongst threads is essential to protecting data from being corrupted. Two common situations resulting in data corruption are when:

- two threads are simultaneously writing to the same database, resulting in a database that contains garbage information with content from both threads

- one thread is writing to a database at a time when another thread is reading from the database, resulting in incorrect information being read.

As illustrated by the two situations, because data corruption occurs when a resource is being accessed by more than one thread during data-writing, there is a need to allow writing threads the opportunity to be the sole occupants of the resource. Only when the writing thread is the only thread accessing the resource can the thread write data accurately, thereby avoiding data corruption.

This need is fulfilled from a technical aspect. In the HLS, any function call that writes to a database utilizes a feature within the Java library that allows a thread to obtain sole possession of the resource by adding the keyword “synchronized” in the method definition [35]. Listing 3.1 illustrates such a method definition.
public synchronized boolean DeleteApplication(String InputApplicationNameToDelete) {
    PackageApplicationsManagerAgent_Application CurrentApplication = null;
    int SizeBeforeRemoval = this.size();
    for(int i = 0; i < this.size(); i++) {
        CurrentApplication = PackageApplicationsManagerAgent_Application.this.get(i);
        if (CurrentApplication.GetApplicationName().equals(InputApplicationNameToDelete)) {
            this.remove(i);
            return false;
        }
    }
    int SizeAfterRemoval = this.size();
    if (SizeAfterRemoval < SizeBeforeRemoval) {
        return true;
    } else {
        return false;
    }
}

Listing 3.1 Example of synchronized method.

If the desired resource is already occupied, the thread executing this function will wait for the resource to be released. During this waiting period, no code within the function definition, which contains the writing procedures, is executed. Upon the release of the resource, the thread will occupy the resource by itself and begin writing new content to the database. When this is done, the thread will release the resource to allow other threads the opportunity to access it.
3.1.1.3 Management of Killing Event Threads

In the HLS, because each event thread is responsible for the execution of a task, the most common scenario for the killing of a thread is when the task has been completed, therefore resulting in no consequences. However, the more uncommon scenario of a system shutdown poses a serious consequence. This consequence will be discussed in a later section as it pertains to backing up vital system information.

3.1.2 Handling Internal Messages

In Section 2.2.2, internal messages were discussed and explained how they would be implemented as events. However, at the time, the other available options were not explored and there was no discussion on why events were preferred over those other options. The focus was on understanding the key concepts that governed the Holonic Logistics System. Now that those concepts have been established, those explanations will now be provided.

One alternative being considered as a method for internal messages was the UDP protocol. By allowing each holonic agent to talk to other holonic agents via UDP, the biggest advantage would be that the agents would be completely loosely-connected, as all that connected the agents to one another was a virtual bridge in the holon. Figure 3.3 shows this virtual bridge.
UDP involves the sending of datagram packets into the network space without the need for an established socket. This means that an agent, as long as there is a port that is established for listening, can receive datagram packets from other agents without having a direct connection with those other agents. As a result, a "virtual bridge" indicates a path from one agent to another, without there being an actual specified path.

By having the agents completely separated from one another, the HLS is given a lot of flexibility in that agents are able to be easily transported from one holon to another. However, there is one significant disadvantage to consider: the available I/O ports available on a holon. Presently, the Holonic Logistics System would require 12 input ports since there are 12 holonic agents. However, those input ports are unique for each agent of that type. If another 100 holonic agents are developed, regardless of whether they are core agents or proprietary agents, the available number of ports will diminish rather quickly. The number of system threads within the HLS would also increase significantly as each port would require a thread to constantly be listening for incoming packets.
UDP datagrams. This could lead to a very inefficient and slow system. Although creating the internal message architecture based on UDP would provide a lot of flexibility, events were used instead as the foundation of the architecture. Events offer the system enough flexibility and more importantly, enough efficiency and accuracy without having to overburden and potentially drain all the hardware resources.

Although the use of events has been chosen as the internal messaging mechanism, there is still a need to properly define the architecture that is in place for the events. Figure 3.4 shows how this architecture is incorporated into the mechanism. All internal messages are packaged into the same type of object, storing three specific pieces of information: destination agent, purpose of message, and information for the destination agent.

The benefit of having a common internal message object is that only one common event function event needs to be called every time a message is sent internally. The alternative would be to pass in individual parameters specific to the event, which would result in a great loss of flexibility for scalability, a waste of memory, and the creation of multiple event functions.
The HLS does not utilize Remote Method Invocation (RMI) or Common Object Request Broker Architecture (CORBA) to perform messaging. RMI is not used because it is not able to propagate synchronization operations such as wait operations and is only able to operate with Java systems. CORBA is not used because it does not support the transfer of byte code. Furthermore, these two technologies possess too difficult a learning curve for new developers to grasp.
3.1.3 Backing Up External Messages

Figure 3.5 shows the UML class diagram for the Messenger Agent.

![UML class diagram for the Messenger Agent](image)

Figure 3.5 Architecture of Messenger Agent.

Because UDP was being used as the protocol for transporting external messages, there was no mechanism that was responsible for tracking the messages. This meant that if Node A was to send an external message to Node B, Node A would have no clue whether or not the message arrived at the destination. The onus was on the Holonic Logistics System to keep track of messages within the holonic environment.

To do this, as highlighted in the Messenger Agent’s class diagram, a backup mechanism was implemented in the output subsystem of the holonic agent. There are two
participants in this mechanism: the sender holon and the receiver holon. The role of the
sender holon will be first explained.

Whenever an external message is sent, a copy of the message is placed into a backup
output queue. When the copy is successfully placed into the queue, a one-minute timer is
started; every message inserted into the queue will have an associated timer. When the
timer expires, an event is triggered and a message is removed from the front of the
backup queue. This newly-retrieved message is then placed into the main output queue
and resent to the destination. Of importance is that before the message is placed into the
main output queue, a copy of the message is created and placed into the end of the
backup queue, after which a new timer is restarted. The purpose of this portion of the
backup mechanism is to keep resending the message every minute until there is a
confirmation from the receiving holon that the message has been successfully received.
Figure 3.6 summarizes the sender portion of the backup mechanism.

The role of the sender guarantees that if an error has occurred in the form of a lost packet
or a dropped packet, the message will be resent. The role of the receiver holon is much
simpler. When a message has been successfully received, the first thing a receiver holon
does is send a confirmation message to the sender holon. Included in this confirmation
message are the message’s identification tag and the receiver holon’s IP-address.
When the initial sender holon receives a message confirming successful reception of the message, the sender holon will remove the message associated with the identification tag from the backup output queue. The following UML sequence diagram in Figure 3.7 summarizes the steps taken by the receiver and the sender upon a successfully received message.

Figure 3.6 Logic flow of message backup mechanism.

Figure 3.7 Logic flow of send-reply protocol between two holons.
3.1.4 Data Logging

The Holonic Logistics Platform is a distributed system meaning that each holon contains a portion of the total network information. As a result of each holon being a vital component of the network database, the HLS requires a data logging mechanism that records information such as: contents of incoming and outgoing messages, contents of user inputs and contents of system events.

To do this, a simple logging mechanism has to be implemented into the system. Every time a message is received and every time a message is sent, the message’s contents and details are logged. Similarly, the HLS also records user inputs into log files each time a user inputs a command. Figure 3.8 provides an example of a log file of message contents and Figure 3.9 provides an example of a log file of user inputs.

/*** New Entry ***/

Date of entry: 24/4/2006
Time of entry: 10:59:49
Message Content:
192.168.0.102<HOLON>192.168.0.103<HOLON>Intelligence<HOLON>l
10-007<HOLON>Demonstration Application<HOLON>
Demonstration Virtual Network<HOLON>Version 1.0<HOLON>
13rf65<HOLON>500<HOLON>500<HOLON>Test;192.168.0.100<HOLON><H
OLON>12rtyui98

Figure 3.8 Example of log-file of incoming external messages.
3.1.5 Persistence Management

Section 2.3.5 discussed the sequence of steps involved in saving vital information prior to a system shutdown and the sequence of steps involved in reloading vital information during system initialization. This section entails the architectural requirements and the technical mechanisms that are involved with those steps. To begin, Figure 3.10 shows the UML class diagram of the Database Agent.

There are two available schemes that could be used to manage the saving and restoration of system information. The first method is called the nested snapshots method. Figure 3.11 provides a simple example of how a nested snapshot works.
Figure 3.10 Architecture of Database Agent.

Figure 3.11 Example of usage of nested snapshots.
With reference to example of the car drawing, in saving the drawing using the nested snapshots method, each component has a designated object, known as a snapshot, that will store all the component's vital information. Furthermore, the snapshot will also be able to store other snapshots.

The snapshot tree shows that the highest level is the snapshot of the car drawing. This snapshot may contain vital information such as the paper which the drawing is performed on and the scale that the drawing is using. The snapshot, however, will not contain information about the actual components of the drawing because those components will have their own snapshots. The snapshot though will store the snapshots of the door, the tires and the trunk. To exemplify another snapshot, the snapshot of the door may contain information such as the colour, height, and width of the door. Once again, even though the door contains a snapshot of the hole, it does not contain any vital information pertaining to the hole. That information is stored in the snapshot of the hole. Therefore, such a method of storing components is in fact very organised as the hierarchy of the objects can be easily restored based on the structure of the snapshot tree.

Initially, the persistence management of the HLS was to be modelled after nested snapshots. Highlighted in the previous Figure 3.10 are the objects that were created to contain backup information specific to each of the holonic agents within the Holonic Logistics System. These objects were designed to be the snapshots of the system. However, there were several drawbacks to this method of persistence. In utilizing the nested snapshot method, there was a need for the system to understand the exact class
structure of the entire system, from the highest level structure of the holon to the lowest structure of the algorithm variable structure. This also meant that as changes were made to the system, that exact class structure would have to be changed, thereby resulting in a large amount of maintenance work created if the system was to be overhauled. Although this method worked well, the potential for making mistakes during the determination of the system class structure and the lack of flexibility to large scale changes was not acceptable for such a system. As a result, another persistence scheme had to be used – one where there was no need to determine the system hierarchy prior to saving system information and one where there was enough flexibility to cope with large-scale changes.

Object serialization is the process of saving an object’s state to a sequence of bytes, as well as the process of rebuilding those bytes into a live object at some future time [36]. This persistence scheme is offered by Java in the form of a library that is able to serialize vital system information into byte code, and this scheme was employed in the Holonic Logistics System. The benefit of this scheme is that once all the classes have been configured to be serializable, meaning that the objects can be converted into byte codes, then the HLS will be able to save and restore the system without the need to know the system class structure. Furthermore, because there is no need to have an imprint of the system class structure, there is a lot of flexibility in allowing the system to be changed without having to undergo unnecessary hierarchical manipulation. Several modifications had to be made to each existing class to allow for the integration of this persistence scheme into the HLS. First, each object class in the HLS had to inherit or implement the Java “Serializable” interface.
Because the next requirement differs depending on whether the process is a save procedure or whether the process is a restore procedure, the former will first be explained. An output file stream is created first to specify which file the bytes will be saved to. Then, the object to be saved is written to that file stream via a newly-created object output stream. Listing 3.2 exemplifies how this simple save-procedure is performed. In this example, the Administration Agent is being saved to the file “AdministrationAgent.ser”.

When restoring a database from a file, an instance of the object has to first be created. After this is done, the newly-created instance will initiated with data read in from a file input stream via an object input stream. Listing 3.3 provides an example of how this is accomplished. In this example, the Administration Agent is being restored from the file “AdministrationAgent.ser”.

Listing 3.2 Example of backing up of an agent.
**Listing 3.3 Example of restoration of an agent.**

```java
1 PackageAdministrationAgent_AdministrationAgent
   AdministrationAgent = null;
2 
3 /***************************/
4 /* Administration Agent */
5 /***************************/
6 
7 Filename = CurrentDirectory + 
8 "//CoreDevelopment//Package_DatabaseAgent 
9 //Holonic_System_Backup_Files// 
   AdministrationAgent.ser";
10 
11 try
12 { 
13   File_InputStream = new File_InputStream(Filename);
14   
15   Object_InputStream = new
16     Object_InputStream(File_InputStream);
17   AdministrationAgent =
18     (PackageAdministrationAgent_
19       AdministrationAgent)
20       Object_InputStream.readObject();
21   
22   Object_InputStream.close();
23 }
24
25 catch(IOException e)
26 { 
27 
28 
29 catch(ClassNotFoundException e)
30 { 
31 
32 }
```
3.1.6 Software Transference

In Section 2.3.8, the transfer of holonic software was discussed. To summarize, software transfer within the holonic environment involves the tasks of adding new software and maintaining existing software. There are also two different types of software: holonic agent software and holonic application software. The UML sequence diagrams in the abovementioned section illustrated the addition and the maintenance of both agent software, as well as the addition of application software into a holon. In the former example, one key point was purposely ignored because the emphasis at that junction of the document was to gain an understanding of the HLS, instead of understanding the intricate details of each system subcomponent. The key point was that if the agent did not exist on the holon, then there was no way to perform any software addition or software maintenance to the agent package. Recall that these tasks of adding and maintaining software are done via the comparison of a listing of software versions that exists on each holon. If the agent package does not exist on that listing, then there is no way for the agents to get updated. Figure 3.12 demonstrates this issue.

Before the explanation of the solution is provided, an understanding must first be established about the current mechanism. The current mechanism is designed to upgrade agent software, and the Holonic Logistics System will continue to use this mechanism because it performs the upgrade on agents that are meant to exist on specific holons. The solution to the problem should only provide holonic users with the capability to transport agents from one holon to another.
A vital point however, is that the HLS may also be a holonic user. If the holonic user is able to determine which agents should exist on which holons, then the present software transfer mechanism will ensure that the software is maintained properly afterwards.

There are two pieces of information that have to be transferred to a new holon when an agent is being transported: the agent package and the communications setup associated with the agent. Transferring the agent package is a simple task that involves the transferring of files containing the agent source code. However, transferring the communications setup is not as simple. Code is kept in several different files that govern the communications switch mechanism within the Holonic Logistic System. As a result,
a copy of only the agent-related code will have to be made and transferred to the new
holon. Figure 3.13 shows how agent-code for each source file will be contained within
one transferred file. Figure 3.14 shows how code is updated on the new holon. Listing
3.4 to Listing 3.10 show the code that performs the update on the holon. Figure 3.15
shows the mechanism of adding an agent from start to finish. Listing 3.11 to Listing 3.17
show the code that adds a new agent.

Start Of Communications Setup File

/* Communications File 1 */
{
    ...agent code extracted from this file
}

/* Communications File 2 */
{
    ...agent code extracted from this file
}

/* Communications File 3 */
{
    ...agent code extracted from this file
}

Figure 3.13 Communications setup file.
Figure 3.14 Logic flow of updating code on a holon.
//step 1: writing byte code to file
try {
    OutputStream os = new FileOutputStream(DirectoryOfClass + ClassName + ".class");

    String[] ComponentsOfReceivedString = ClassByteCodeAsString.split("<COLIN>").length;
    ComponentsOfReceivedString = ClassByteCodeAsString.split("<COLIN>");

    for(int i = 0; i < ComponentsOfReceivedString.length; i++)
    {
        Integer ReceivedStringAsInteger = Integer.valueOf(ComponentsOfReceivedString[i]);
        int ReceivedStringAsInt = ReceivedStringAsInteger.intValue();
        e.printStackTrace();
        System.out.println(e);
        os.write(ReceivedStringAsInt);
    }
}

//step 2: load class to JVM
try {
    //put together class name in package notation (ie. PackageHolonic.HolonicMethod)
}

Listing 3.4 Part 1 of code that updates agent code.
35  String ClassPathName = PackageBeingAccessed + "." + ClassName;
36  //load class
37  ClassLoader Loader =
   ClassLoader.getSystemClassLoader();
38  Class ClassToBeLoaded =
   Loader.loadClass(ClassPathName);
39  //now call function within class to notify holonic system that class has been loaded
40  String[] DummyArgs = new String[] {};
41  try
42  {
43   
44   Method MethodToInvoke =
      ClassToBeLoaded.getMethod("TestIfSoftwareHasBeenTransferredSuccessfully", null);
45  
46  try
47  {
48   
49   MethodToInvoke.invoke(null, (Object[])
      DummyArgs);
50  
51   
52   catch (IllegalArgumentException e)
53   {
54     e.printStackTrace();
55   }
56   
57   catch (IllegalAccessException e)
58   {
59     e.printStackTrace();
60   }
61   
62   catch (InvocationTargetException e)
63   {
64     e.printStackTrace();
65   }
66   
67   
68   catch (SecurityException e)
69   {
70   
    Listing 3.5 Part 2 of code that updates agent code.
false;

NoSuchMethodException e)
  
System.out.println(e);
  
System.out.println(e);
  }
}

catch(ClassNotFoundException e)
  
{
  
}

//step 3: update software version file

try
  
{
  
  //retrieve file to read from (software version file)
  
  CurrentDirectory = System.getProperty("user.dir");
  
  String ReadFilePath = CurrentDirectory + HI/Core
Development//Package SoftwareTransferAgent//"Software_Version_Files//Holonic Software Versions.txt";

  
  //retrieve file to write to (temp software version file)
  
  String WriteFilePath = CurrentDirectory + "//Core Development//Package_SoftwareTransferAgent//"

Listing 3.6 Part 3 of code that updates agent code.
//read text file and write every line to the temp text file.
//then, when we get to the class that is to be update, write new data.
BufferedReader in = new BufferedReader(new FileReader(ReadFilePath));
BufferedWriter out = new BufferedWriter(new FileWriter(WriteFilePath, false));

String CurrentLineBeingRead = "";
while((CurrentLineBeingRead = in.readLine()) != null)
{
    if(CurrentLineBeingRead.substring(0,4).equals("/**"))
    {
        out.write(CurrentLineBeingRead);
        out.newLine();
    }
    else
    {
        String[] ComponentsOfClass = new String[CurrentLineBeingRead.split("-").length];
        ComponentsOfClass = CurrentLineBeingRead.split("-");

        if(ClassName.equals(ComponentsOfClass[0]))
        {
            ClassVersionNumber = ComponentsOfClass[1];
            out.write(ClassName + ":" + ClassVersionNumber);
            out.newLine();
            ClassExistsInSoftwareVersionFile = true;
        }
        else
        {
            Listing 3.7 Part 4 of code that updates agent code.
134 {
135    out.write(CurrentLineBeingRead);
136    out.newLine();
137 }
138 }
139 }
140 out.close();
142 }
143 catch (IOException e)
144 {
145
146 }
147
148
149
150 /*
151 * part b: copy over the temp text file and if class is new, then
152 * write to the software version text file
153 */
154 try
155 {
156 // retrieve file to read from (temp software version file)
157 CurrentDirectory = System.getProperty("user.dir");
158
159 String ReadFilePath = CurrentDirectory + "/Core Development//Package_SoftwareTransferAgent//"
160 + "Software_Version_Files//Temp Holonic Software Versions.txt";
161
162 // retrieve file to write to (software version file)
163 String WriteFilePath = CurrentDirectory + "/Core Development//Package_SoftwareTransferAgent//"
164 + "Software_Version_Files//Holonic Software Versions.txt";

Listing 3.8 Part 5 of code that updates agent code.
166
167 //read text file and write every line to the temp
text file.
168 //then, when we get to the class that is to be
update, write new data.
169 BufferedReader in = new BufferedReader(new
FileReader(ReadFilePath));
170 BufferedWriter out = new BufferedWriter(new
FileWriter(WriteFilePath, false));
171
172 String CurrentLineBeingRead = "";
173 while((CurrentLineBeingRead = in.readLine()) !=
null)
174 {
175 if(CurrentLineBeingRead.substring
(0,4).equals("/***/))
176 {
177 //retrieving title being read in file
178 int IndexOfEndingStars =
CurrentLineBeingRead.indexOf("***/");
179 String TextBeingRead =
CurrentLineBeingRead.substring(5,IndexOfEndingS
stars-1);
180
181 //retrieving title being read in class name
182 int IndexOf_ = ClassName.indexOf("_");
183 String SystemName = ClassName.substring(0,
IndexOf_);
184 String ClassPortionBeingRead =
SystemName.substring(0,7) +
"_" + SystemName.substring(7);
185
186 if((TextBeingRead.equals(ClassPortionBeingRead))
187 && !ClassExistsInSoftwareVersionFile)
188 {
189     out.write(CurrentLineBeingRead);
190     out.newLine();
191     out.write(ClassName + "_" +
ClassVersionNumber);
192     out.newLine();
193 }

Listing 3.9 Part 6 of code that updates agent code.
Listing 3.10 Part 7 of code that updates agent code.
Request permission to begin software transfer mechanism

If Holon B granted permission to begin software transfer mechanism, Holon A will send the source files associated with the holonic agent. If not, then Holon A will continue to request for permission every 60 seconds until permission is granted or until Holon A decides to stop.

Check to ensure that newly-downloaded files are properly loaded into system. Update own list of source files with new updated version numbers. If new files were downloaded, add source file information into list.

Send confirmation to Holon A that agent source files have been successfully transferred and added to Holon B’s system. Holon B is now waiting for communications setup file.

Send communications setup file

Add communications setup code to appropriate communications files.

Inform Holon A that it has completed the software transfer mechanism

Figure 3.15 Logic flow of adding an agent to a holon.
ExecuteHolonicTechnologyPlatform LaunchPlatform =
            Event.GetHolonicTechnologyPlatformMaster().GetLaunchingPlatform();

/*
 * PART 1: run agent addition code
 */
***********************************/
/* step 1: retrieve data from file */
***********************************/
String CurrentDirectory = System.getProperty("user.dir");
String FilePath = CurrentDirectory
+ "/Core Development//Package_SoftwareTransferAgent//" + "Software_Version_Files//AgentToBeAdded.txt";
String CumulativeData = new String(""");
try {
    BufferedReader in = new BufferedReader(new FileReader(FilePath));
    String CurrentLineBeingRead = "";
    while ( (CurrentLineBeingRead = in.readLine()) != null) {
        CumulativeData = CumulativeData + CurrentLineBeingRead;
    }
}
catch (IOException e) {

/***********************************/
/* step 2: separate data and create new files under new directory */
***********************************/
String[] CumulativeData.split("<AGTTX>").length;
String[] AgentPackageByteCodeArray = CumulativeData.split("<AGTTX>");

Listing 3.11 Part 1 of code that adds new agent.
45 String AgentType = AgentPackageByteCodeArray[1];
46
47 String AgentName = AgentPackageByteCodeArray[2];
48 String[] ComponentsOfSoftwareUpdate = new
        String[AgentName.split(" ").length];
49 ComponentsOfSoftwareUpdate = AgentName.split(" ");
50 String ConnectedAgentName =
        ComponentsOfSoftwareUpdate[0] +
        ComponentsOfSoftwareUpdate[1];
51
52 String AgentPackageName = AgentPackageByteCodeArray[3];
53 String AgentPackageByteCode =
        AgentPackageByteCodeArray[4];
54
55 // extract entire byte code information of individual
56 // classes
57 String[] IndividualClassByteCodeArray = new
        String[AgentPackageByteCode.split("<AGTCCLASS>").
        length];
58 IndividualClassByteCodeArray =
        AgentPackageByteCode.split("<AGTCCLASS>");
59
60 // step ii: create directory in agent development
61 File NewAgentDirectory = null;
62
63 NewAgentDirectory = new File(CurrentDirectory
64 + "/bin//Agents//" + AgentPackageName);
65
66 boolean DirectoryExists = NewAgentDirectory.exists();
67 if (DirectoryExists)
68 {  
69 }
70 else  
71 {  
72 NewAgentDirectory.mkdir();
73 }
74 }
75
76 Listing 3.12 Part 2 of code that adds new agent.
81
82
83 LinkedList ListOfFilenames = new LinkedList();
84 LinkedList ListOfVersionNumbers = new LinkedList();
85
86 // step iii: for each class, write the byte code to the file
87 for(int i = 0; i < IndividualClassByteCodeArray.length; i++)
88 {
89     String CurrentClassInformation =
90         IndividualClassByteCodeArray[i];
91
92     // for each individual class, extract class name and byte code.
93     String[] ClassInformationArray = new String[CurrentClassInformation.split("<SW>").length];
94     ClassInformationArray =
95         CurrentClassInformation.split("<SW>");
96
97     String ClassName = ClassInformationArray[0];
98     String ClassVersionNumber = ClassInformationArray[1];
99     String ClassByteCode = ClassInformationArray[2];
100    ListOfFilenames.add(ClassName);
101    ListOfVersionNumbers.add(ClassVersionNumber);
102
103     try
104         {
105             OutputStream os = null;
106             os = new FileOutputStream(
107                 CurrentDirectory + "//bin//Agents//" +
108                 AgentPackageName + "//" +
109                 ClassName + ".class");
110         }
111
112         ComponentsOfReceivedString =
113             ClassByteCode.split("<COLIN>");
114     double MaxRead = 128.0;
115     byte[] WriteArray = new byte[(int)MaxRead];
116     int IndexTracker = 0;

Listing 3.13 Part 3 of code that adds new agent.
for(int l = 0; l < ComponentsOfReceivedString.length; l++)
{
    Integer ReceivedStringAsInteger =
        Integer.valueOf(ComponentsOfReceivedString[l]);
    int ReceivedStringAsInt =
        ReceivedStringAsInteger.intValue();
    try
    {
        os.write(ReceivedStringAsInt);
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
}

Listing 3.14 Part 4 of code that adds new agent.
for(int i = 0; i < ListOfFilenames.size(); i++)
    String FilenameToWrite = (String)ListOfFilenames.get(i);
    String VersionNumberToWrite = (String)ListOfVersionNumbers.get(i);
    Out.write(FilenameToWrite + "-" + VersionNumberToWrite);
    Out.newLine();
    Out.close();
    }
    catch(IOException e)
    {
    
    
    
    
    /************************************************/
    
    /* step 3: instantiate agent and add it to list of agents in htp */
    
    **************************************************************************/
    
    /*
    * this is to be done manually upon rebooting of system
    */
    
    char DoubleQuotationCharacter = '"';
    String DoubleQuotationCharacterAsString = new Character((char)DoubleQuotationCharacter).toString();

Listing 3.15 Part 5 of code that adds new agent.
Listing 3.16 Part 6 of code that adds new agent.
DoubleQuotationCharacter = '\"';

DoubleQuotationCharacterAsString = new
    Character((char) DoubleQuotationCharacter).toString();

String Command = DoubleQuotationCharacterAsString
    + ClassDirectory + DoubleQuotationCharacterAsString;

try {
    Process RunAddNewAgent =
        Runtime.getRuntime().exec(Command);
} catch (IOException e) {
    System.out.println(e);
}

//let the thread sleep while the batch file is being run
long SleepTime = 15000;
try {
    Thread.sleep(SleepTime);
} catch (InterruptedException e) {
    e.printStackTrace();
}

/*************************************/
/* step 8: relaunch HTP */
/*************************************/
LaunchPlatform.FireLaunchNewHTPEvent();
3.1.7 Algorithms Management

Figure 3.16 shows the UML class diagram of the Algorithms Agent.

When an algorithm has to be added to the Holonic Logistics System, the code for the algorithm class is placed in a folder designated for algorithms. There are several requirements that an algorithm class must conform to, as listed below.
1. Each algorithm class must implement the Serializable interface. By implementing the Serializable interface, the HLS would be able to save states of the algorithm when the system is being backed up.

2. Each algorithm class must have a constructor. Although this may seem trivial, a constructor will ensure that the algorithm begins with the proper values and environment settings.

3. Each algorithm class must have a ProcessData method. This method is common to all algorithm classes. This method is necessary to keep running the algorithm until it stops.

4. Each algorithm class must have an InitiateAlgorithm method. This method is common to all algorithm classes. This method is necessary to initiate the running of the algorithm within the holonic environment.

5. Each algorithm class must have a boolean instance called RunAlgorithm. This variable keeps track of whether the algorithm should be kept running.

6. Each algorithm class must have a PackageAlgorithmsAgent - InitiateAlgorithmEvent instance called InstanceOfEvent. This event stores information required for the algorithm to be initiated.
In addition to the above mentioned six items, three additional details have to be attended to. First, an instance of the algorithm must be instantiated in PackageAlgorithmsAgent-QueueOfAlgorithmsForProcessing. This ensures that the algorithm is listed in the queue of algorithms. Second, an accessor method must be created to allow the retrieval of the algorithm in PackageAlgorithmsAgent-QueueOfAlgorithmsForProcessing. Third, in PackageAlgorithmsAgent-InitiateAlgorithmEventHandler, within the method InitiateAlgorithmEventReceived, code must be added to allow the execution and continuous running of an algorithm.

Listing 3.18 and Listing 3.19 show an algorithm class that incorporates the required six items. Listing 3.20 and Listing 3.21 show the two required items in PackageAlgorithmsAgent-QueueOfAlgorithmsForProcessing. Listing 3.22 and Listing 3.23 show the added code in PackageAlgorithmsAgent_InitiateAlgorithmEventHandler.
package Package_AlgorithmsAgent.Algorithms;
import java.io.Serializable;
import java.util.Date;
import java.util.Timer;
import java.util.TimerTask;
import Package_AbstractComponents.*;
import Package_AlgorithmsAgent.*;
import package_AIgorithmsAgent.*;
import Package_VirtualNetworksManagerAgent.*;
public class Algorithm_SelectingLeader implements Serializable

{  
  import Package_AlgorithmsAgent.InitiateAIgorithmEvent;
  import Package_AlgorithmsAgent.Variable;

  public Algorithm_SelectingLeader()
  { 
    RunAlgorithm = false;
    InstanceOfEvent = null;
  }

  Listing 3.18 Part 1 of Algorithm Class.

  146
public String GetNodeStatus()
{
    return NodeStatus;
}

public void ProcessData(
    String InputDataToProcess,
    PackageAlgorithmsAgent_InitiateAlgorithmEvent InputlnitiateAlgorithmEvent)
{
    <insert code required to run algorithm>
}

public void InitiateAlgorithm(
    PackageAlgorithmsAgent_InitiateAlgorithmEvent
    InputInitiateAlgorithmEvent)
{
    <insert code required to initiate algorithm>
}
package Package_AlgorithmsAgent;
import java.io.Serializable;
import java.util.*;
import Package_AlgorithmsAgent.Algorithms.*;

public class PackageAlgorithmsAgent_QueueOfAlgorithmsForProcessing
    extends LinkedList implements Serializable
{
    private Algorithm_SelectingLeader SelectingLeaderAlgorithm;

    SelectingLeaderAlgorithm = new Algorithm_SelectingLeader();
    this.add(SelectingLeaderAlgorithm);
}

Listing 3.20 Required item 1.
public Algorithm SelectingLeader
    GetSelectingLeaderAlgorithm()
{
    return SelectingLeaderAlgorithm;
}

Listing 3.21 Required item 2.

package Package_AlgorithmsAgent;
public class PackageAlgorithmsAgent Initiate
    AlgorithmEventHandler implements
    PackageAlgorithmsAgent
    InitiateAlgorithmEventListener
{
    String AlgorithmToAccess =
        Event.GetAlgorithmToAccess();
    String PurposeOfEvent =
        Event.GetPurposeOfEvent();
    if (PurposeOfEvent.equals("Start")
    {
        if (AlgorithmToAccess.equals("SelectingLeader")
        {
            Event.GetQueueOfAlgorithmsForProcessing().
                GetSelectingLeaderAlgorithm().InitiateAlgorithm(Event);
        }
    }
}

Listing 3.22 Code initiating algorithm.
//the event is requesting that the system evaluate the
message based on the algorithm
else if (PurposeOfEvent.equals("Evaluate"))
{
if (AlgorithmToAccess.equals("SelectingLeaderAlgorithm"))
{
String DataToProcess = Event.GetDataToProcess();
Event.GetQueueOfAlgorithmsForProcessing()
.GetSelectingLeaderAlgorithm().ProcessData(DataToProcess, Event);
}
}

Listing 3.23 Code performing algorithm evaluation.

3.1.8 Development Tools

3.1.8.1 Agent Generator and Holonic Application Generator

As Section 2.3.6 explained, the Development Agent allows agents and holonic applications to be created while the system is running. The agent development tool generates fundamental agent code based on the agent template, while the holonic application development tool generates fundamental holonic application code based on the holonic application template. Figure 3.17 is a UML sequence-diagram that demonstrates the sequence of steps taken to create the agent code, whereas Listing 3.24, Listing 3.25 and Listing 3.26 are examples of the actual generated code. Figure 3.18 is a UML sequence-diagram that demonstrates the sequence of steps taken to create the holonic application code. Because there are too many generated files, the code of those files will not be shown.
Create new directory for agent code

Development Agent

Create agent source file

Create agent database

master source file

Windows File Directory Manager

Figure 3.17 Logic flow of agent creation.
package Agents.SmithAgentTeam;

public class BobbyAgent extends PackageAbstractComponents_AbstractAgent {

private int VersionNumber;
private BobbyAgent_DatabaseMaster DatabaseMaster;

public BobbyAgent() {
    VersionNumber = 1;
    DatabaseMaster = new BobbyAgent_DatabaseMaster();
    /* CHANGE SINGLE QUOTES TO DOUBLE QUOTES!!! */
    this.SetAgentName("Bobby Agent");
}

public BobbyAgent_DatabaseMaster GetDatabaseMaster() {
    return DatabaseMaster;
}

Listing 3.24 Part 1 of generated agent code.
```java
public static void TestIfSoftwareHasBeenTransferredSuccessfully()
{
    System.out.println("SYSTEM INFORMATION: The following class, BobbyAgent, has been loaded successfully after the transfer of byte codes.");
}
```

Listing 3.25 Part 2 of generated agent code.
```java
public static void TestIfSoftwareHasBeenTransferredSuccessfully()
{
    System.out.println("SYSTEM INFORMATION: The following class, BobbyAgent.DatabaseMaster, has been loaded successfully after the transfer of byte codes.");
}
```

Listing 3.26 Part 3 of generated agent code.

Figure 3.18 Logic flow of holonic application creation.
3.1.8.2 File Replication and Transferability Tester

The remaining two development tools are developed primarily for testing. Those tools are the file replication tool and the transferability tester. The file replication tool copies a desired file and renames it to another class name. Upon creation, a method will be used to run the class to ensure that the class is functioning as expected. These two features within the tool make it more convenient for a developer to implement new files while the system is running. Listing 3.27 and Listing 3.28 provide the code used to copy and rename a new file. Listing 3.29 provides the code used to run a class.

```java
1 String PackageOfExistingFile = 
    Event.GetPackageOfExistingFileBeingAccessed();
2 String ExistingFilename = Event.GetExistingFilename();
3 String PackageOfNewFile = 
4 String NewFilename = Event.GetNewFilename();
5 String CurrentDirectory = System.getProperty("user.dir");
67
1* initialise input stream
9 */
10 InputStream is = null;
11 try
12 { 
13    is = new FileInputStream(CurrentDirectory + "/" +
14        PackageOfExistingFile
15        + "/" + ExistingFilename + ".class");
16 }
17 catch (FileNotFoundException e)
18 
19    e.printStackTrace();
20 }
21
22
23 */
24 * initialise output stream
25 */
26 OutputStream os = null;
27 try
28 {

Listing 3.27 Part 1 of code used to copy and rename a file.

155
30 os = new FileOutputStream(CurrentDirectory + "//Core" +
     PackageOfNewFile + "//" + NewFilename + ".class");  
32 }  
33 catch (FileNotFoundException e)  
34 {  
35 e.printStackTrace();  
36 }  
37  
38 * initialise class info  
39 */  
40 ClassInfo classInfo = new ClassInfo();  
41 try  
42 {  
43 new ClassFileReader().read(is, classInfo);  
44 }  
45 catch (ClassFileParseException e)  
46 {  
47 e.printStackTrace();  
48 }  
49  
50 */  
51 catch (IOException e)  
52 {  
53 e.printStackTrace();  
54 }  
55 classInfo.setName(NewFilename); // Java requires the class  
56 name to match the file name;  
57  
58 */  
59 */  
60 * initialise class file writer  
61 */  
62  
63 try  
64 {  
65 new ClassFileWriter().write(classInfo, os);  
66 }  
67 catch (IOException e)  
68 {  
69 e.printStackTrace();  
70 }  
71 catch (IOException e)  
72 {  
73 e.printStackTrace();  
74 }  
75  
76 */  
77 */  
78 * close input and output stream  
79 */

Listing 3.28 Part 2 of code used to copy and rename a file.
try
{
  //put together class name in package notation (ie. PackageHolonic.HolonicClass)
  String ClassPathName = PackageBeingAccessed + "." + Filename;

  System.out.println("DEVELOPER INFORMATION:
      Notation for class-loading is '" +
      ClassPathName + ").");

  //load class
  ClassLoader Loader = ClassLoader.getSystemClassLoader();
  Class ClassToBeLoaded = Loader.loadClass(ClassPathName);

  //now call function within class to notify holonic system
  //that class has been loaded
  String[] DummyArgs = new String[]{};
  try
  {
    Method MethodToInvoke =
    ClassToBeLoaded.getMethod("TestIfSoftwareHasBeen
      NotransferredSuccessfully", null);

    try
    {
      MethodToInvoke.invoke(null, (Object[]) DummyArgs);
    }
    catch (IllegalArgumentException e)
    {
      e.printStackTrace();
    }
    catch (IllegalAccessException e)
    {
      e.printStackTrace();
    }
    catch (InvocationTargetException e)
    {
      e.printStackTrace();
    }

    catch (SecurityException e)
    {
      System.out.println(e);
    }
    catch (NoSuchMethodException e)
    {
      System.out.println(e);
    }

    catch(ClassNotFoundException e)
    {System.out.println(e);
  }

  Listing 3.29 Code Used to Run Renamed Class-File.
The transferability tester is a development tool that tests whether or not the file has been successfully downloaded from another holon. Although this is already done automatically upon the completion of a software transfer, a developer has the option of performing the test again if the results of the test were missed. Listing 3.30 is the code for running the test.

```java
try {
    Method MethodToInvoke = ClassToBeLoaded.getMethod
        ("TestIfSoftwareHasBeenTransferredSuccessfully", null);
    try {
        MethodToInvoke.invoke(null, (Object[]) DummyArgs);
    } catch (IllegalArgumentException e) {
        e.printStackTrace();
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    } catch (InvocationTargetException e) {
        e.printStackTrace();
    } catch (SecurityException e) {
        System.out.println(e);
    } catch (NoSuchMethodException e) {
        System.out.println(e);
    }
}
```

Listing 3.30 Generic Code Used To Test a Class File.
3.1.9 Yellow Pages versus White Pages

The Holonic Logistics System utilizes the concept of white pages and yellow pages to organize its holonic agents and its holonic applications. White pages provide a list of the agents and applications based on the alphabetical order of the names. Yellow pages provide a list of the agent and applications based on the alphabetical order of the categories to which they belong. Figure 3.19 depicts the difference.

Figure 3.19 White pages vs. yellow pages.

Understanding that this was the way to organize holonic agents and holonic applications when presenting information to the user, the difficulty was in the retrieval and the sorting of the information. The list of agents and the list of holonic applications were stored in different areas of the Holonic Logistics System, and to compound the difficulty, the information pertaining to the holonic applications were embedded within the Applications Manager Agent. A decision had to be made based on how the holonic agent related to the holonic application. The relationship between the two was very weak as a
consequence of the holonic agents within the application communicating with the holonic agents within the HLS, rather than the applications communicating with the agents. Since the relationship was weak, there would be two different options that would be made available to the user. The first option would list the applications in the form of either the yellow pages or the white pages, and the second option would list the agents in the form of either the yellow pages or the white pages.

Implementation would be simple since two options were provided. Whether the list pertained to the holonic applications or whether the list pertained to the holonic agents, the task simply involved traversing through the lists of the objects and outputting the objects in the desired page-format. The UML activity diagram in Figure 3.20 demonstrates the steps taken.
Retrieve list of agents OR retrieve list of applications

Is white pages the desired format?

YES

Sort list into alphabetical order

NO

Determine categories that objects belong to

Sort list based on alphabetical order of category names

Sort category list into alphabetical order

Output list to Holonic Logistics System

Figure 3.20 Logic Flow of Displaying White and Yellow Pages.
3.2 Interface Issues

In developing the interface between the Holonic Logistics System and the layers, both above and below, several obstacles were stumbled upon. The following subsections discuss those obstacles.

3.2.1 Information Coordination

With so much information being exchanged between the different layers of the Holonic Technology Platform and with the traffic of information increasingly exponentially as information is sent throughout the holonic environment, there was a need to be able to interpret the data in an organized fashion. The method of organization decided was to separate the data into levels of categories. Figure 3.21 shows an abstractization of this method.

Figure 3.21 Abstractization of Levels of Information Storage.
As the figure indicates, data is separated into levels with different levels contained within one another. This may seem simple, but is of extreme significance because some levels of data may only relate to a portion of the level above it. For example, the Level 4 data in the figure is only related to the Level 3 data that encapsulates it. This concept is further emphasized by the separations that exist even within the same levels of data. As a result of the encapsulations that occur within the data, the data is better organized, which in return causes fewer problems for the holon on the receiving end of the data.

3.2.2 Layered Communication

This section re-emphasizes the need for proper interfacing between the layers, above and below the HLS, and the HLS itself. Figure 3.22 relates the relationships that exist between the holons and the holonic systems that exist within it.

When a holon is communicating with another holon, the connection shown in Figure 3.23 is created. As mentioned, there is only one virtual connection that is created between two communicating holons via the use of UDP. However, the layers within one holon can still communicate freely with the layers of another holon. The key to this layer-to-layer communication is that a layer can only talk to another holon’s layer if both layers are of the same type. For example, the HLS can only talk to the HLS, and the holonic applications layer can only talk to the holonic applications layer. In this manner, the communications scheme is similar to that of the layered-architecture used as the backbone of the Internet [37].
Figure 3.22 Relationship between holon and internal holonic systems.
Figure 3.23 Inter-holon communications.
The following UML activity diagrams in Figure 3.24 and in Figure 3.25 show how the layers are able to send messages to each other even though the layers exist on different holons. The example that is used is one where the Holonic Applications layer is communicating with another Holonic Applications layer.

![UML diagrams]

Figure 3.24 Information traversing from HSS to HCS.

As shown by the example on the sender side in Figure 3.24, the layers interact with one another via UDP, thereby conforming to the system objectives that all external communication is to be performed using UDP. The example also reveals that each layer has a message analyzer mechanism that is able to determine the purpose of the message.
This message analyzer is of extreme value because as messages flow throughout the network and within the holon, the traffic flow will need to be managed efficiently.

As Figure 3.25 displays, the same checks are performed on the receiving side of the message as on the sending side of the message. The message propagates through each layer via the results of the message analyzer.

Figure 3.25 Information traversing from HCS to HSS.
3.3 Intelligence Issues

In integrating simple intelligence into the Holonic Logistics System, several problems were discovered, as detailed in this subsection.

3.3.1 Locating Node Reference Distance

In a distributed network, the topology of the network is not always known to each individual node. The same fact can be stated about the holons in the distributed environment created by the Holonic Logistics Platform. However, despite not being able to identify the configuration of the network, it is still possible to ascertain the relationship between each holon in the holonic environment. There is a necessity for the means to determine these various relationships.

One relationship that the Holonic Logistics System is able to determine is the network size with respect to a single holon, as shown by Figure 3.26. From the figure, the relationship that exists between the one holon and the other holons in the environment is measured in the metric of hops. This is a vital measurement of the network size because the holon can then factor the size into any of the algorithms that it is running. For instance, if a chosen algorithm is meant for short hop distances, and if the furthest holon is too far away, then another algorithm has to be selected.
The algorithm used to perform the search for the maximum number of hops is a simple one. To begin, the holon wishing to discover this measurement sends an external inquiry message to notify that the search has begun. It is important to note that only holons that have this algorithm within their HLS are able to play a role in this search. For the purpose of this explanation, we will assume that all holons participating have this algorithm installed into their HLS.

Upon receiving the inquiry message, the receiving holons compare the current maximum hop value within their database with the message’s current maximum hop value. If the message’s value is less than the databases value, then the following steps are taken:
1. set value of maximum hop count within database to be the value of the message’s hop count

2. increment the hop count in the message

3. forward the message to neighbours of the receiving holon

4. send a reply to the “home node” (the node initiating the search) with the number of hops that the receiving node is away by.

To keep the algorithm simple, the search will last several minutes. When that time frame has expired, the algorithm will stop, during which time the home node will send out an external message to its neighbours to halt the running of the algorithm. Upon receiving the halt notification, the receiving nodes will simply stop executing the algorithm and forward the halt message to its neighbours. When the home node has sent this external halt notification, it will retrieve the most current maximum hop value and store it into the HLS database. This will make the value available to the different algorithms that may require this information. The following code shown in Listing 3.31 to Listing 3.35 details the algorithm.
/*
 * step 1: retrieve data
 */
String[] SeparatedMessageArray = new String[InputDataToProcess.split(";").length];
SeparatedMessageArray = InputDataToProcess.split(";");
String AlgorithmName = SeparatedMessageArray[0];
String VirtualNetworkToAccess = SeparatedMessageArray[1];
String PurposeOfEvent = SeparatedMessageArray[2];
String AlgorithmSpecificData = SeparatedMessageArray[3];

/*
 * step 2: retrieve data within algorithm specific data
 */
String[] AlgorithmInformationArray = new String[AlgorithmSpecificData.split("<alg>").length];
AlgorithmInformationArray = AlgorithmSpecificData.split("<alg>");

String MessageType = AlgorithmInformationArray[0];
if (MessageType.equals("Outbound")) {
    String IPNumberOfHomeNode = AlgorithmInformationArray[1];
    String MessageHopCount = AlgorithmInformationArray[2];
    int MessageHopCountAsInt = Integer.parseInt(MessageHopCount);
    String PathTravelledInformation = AlgorithmInformationArray[3];
    String[] PathTravelledInformationArray = new String[PathTravelledInformation.split("-").length];
    PathTravelledInformationArray = PathTravelledInformation.split("-");
}

/*
 * step 3: make comparisons now
 */
if (MessageHopCountAsInt < CurrentHopCount) {
    /*
     * step 3a: forward message with updated values to neighbours
     */
    //have to make update of hop count

Listing 3.31 Part 1 of code determining maximum hop number.
CurrentHopCount = MessageHopCountAsInt;
int UpdatedMessageHopCount = MessageHopCountAsInt + 1;

//now that update to hop count has been made, forward message to neighbours
PackageVirtualNetworksManagerAgent.QueueOfVirtualLinks
QueueOfVirtualLinks = InputInitiateAlgorithmEvent.GetQueueOfVirtualLinks();

PackageVirtualNetworksManagerAgent.VirtualLink
CurrentVirtualLink = null;
for(int i = 0; i < QueueOfVirtualLinks.size(); i++) {
    //for each neighbour of the initiating node, send a request to start algorithm
    CurrentVirtualLink =
    (PackageVirtualNetworksManagerAgent.VirtualLink)
    QueueOfVirtualLinks.get(i);
    //now that update to hop count has been made, forward message to neighbours
    PackageVirtualNetworksManagerAgent.QueueOfVirtualLinks
    QueueOfVirtualLinks = InputInitiateAlgorithmEvent.GetQueueOfVirtualLinks();

    String Nadel =
    CurrentVirtualLink.GetAddressOfFirstNodeAsString();
    String Node2 =
    CurrentVirtualLink.GetAddressOfSecondNodeAsString();
    String DestinationAddress = 
    if (Node1.equals(InputInitiateAlgorithmEvent.GetOwnIPNumber()))
    DestinationAddress = Node2;
    else
    DestinationAddress = Node1;

    //check to see if we are sending to the home node
    if (DestinationAddress.equals(IPNumberOfHomeNode))
    { //do nothing because we don't want to forward message to home }
    else
    { //update path travelled
        String UpdatedPathTravelled = " 
        for(int j = 0; j < 
        PathTravelledInformationArray.length; j++)
        { UpdatedPathTravelled =
            UpdatedPathTravelled +
            " 
        }

Listing 3.32 Part 2 of code determining maximum hop number.
Listing 3.33 Part 3 of code determining maximum hop number.
117 //create message
118 String DestinationAddress =
119 PathTravelledInformationArray[PathTravelledInformationArray.length -
120 1];
121
122 PackageAbstractComponents_AbstractMessage NewMessage =
123 InputInitiateAlgorithmEvent.CreateNewMessage(DestinationAddress,
124 VirtualNetworkToAccess, ApplicationData);
125
126 //send message
127 InputInitiateAlgorithmEvent.PlaceAndSendMessageOnOutput
128 Queue(NewMessage);
129 else
130 {
131 //do nothing
132 }
133 }
134
135 else if(MessageType.equals("Home"))
136 {
137 String IPNumberOfHomeNode = AlgorithmInformationArray[1];
138 String HopCountBeingReturned =
139 AlgorithmInformationArray[2];
140 int HopCountBeingReturnedAsInt =
141 Integer.parseInt(HopCountBeingReturned);
142 String ReturnPathInformation =
143 AlgorithmInformationArray[3];
144 String[] ReturnPathInformationArray = new
145 String[ReturnPathInformation.split("-").length];
146 ReturnPathInformationArray =
147 ReturnPathInformation.split("-");
148 //check to see if this node is the home node
149 if(IPNumberOfHomeNode.equals(InputInitiateAlgorithmEvent.GetOwnIPNum
150 ber()))
151 {
152 if(HopCountBeingReturnedAsInt > MaxValue)
153 {
154 MaxValue = HopCountBeingReturnedAsInt;
155 InputInitiateAlgorithmEvent.SetMaxHopCount
156 InNetwork(MaxValue);
157 }
158 }
159 else
160 
161 String ReturnPath = ";
162 for(int i = 0; i < (ReturnPathInformationArray.length -
Listing 3.34 Part 4 of code determining maximum hop number.
ReturnPath = ReturnPath +
    ReturnPathInformationArray[j] + "-";

//create contents of message
String AlgorithmData = "Home" + "<alg>" +
    IPNumberOfHomeNode
+ "<alg>" + HopCountBeingReturned
+ "<alg>" + ReturnPath;

String ApplicationData = "FindingMaximumNumberOfHops" +
    ";" +
    VirtualNetworkToAccess + ";" + "DoNotCareForNow" + ";" +
    AlgorithmData;

//create message
String DestinationAddress =
    ReturnPathInformationArray[ReturnPathIn
    formationArray.length - 1];

PackageAbstractComponents_AbstractMessage NewMessage =
    InputInitiateAlgorithmEvent.CreateNewMessage{
        DestinationAddress,
        VirtualNetworkToAccess, ApplicationData};

//send message
InputInitiateAlgorithmEvent.Place
    AndSendMessageOnOutputQueue(NewMessage);

else if(MessageType.equals("Halt"))
{
    RunAlgorithm = false;
    CurrentHopCount = 99999;
    InputInitiateAlgorithmEvent.SendMessageToConsole("Algorithm
        Name: Finding Maximum
        Number of Hops");
    InputInitiateAlgorithmEvent.SendMessageToConsole("Algorithm
        Status: Completed");
}

Listing 3.35 Part 5 of code determining maximum hop number.
From the code, several key points can be summarized:

- There are three types of messages: outbound, home and halt. Outbound messages signify messages that are traversing the network and inquiring about the hop distances relative to the home node. Home messages signify messages that are notifying the home node of new hop distances. Halt messages identify messages that are traversing throughout the network to halt the running of the algorithm.

- Comparisons of maximum hop values are made at the beginning of the algorithm to determine whether any action should be taken.

- If the message’s value is lower than that of the holon’s, then updates are made to the message’s value and the inquiry message is forwarded. In addition, messages are sent to the home node to identify the hop distances.

3.3.2 Clock Synchronization

One issue that exists on all distributed networks is how the nodes synchronize with one another [39]. Because there is no centralized location that maintains when nodes log off and log onto the network, there is no way to synchronize the nodes as they enter the network. Furthermore, to compound the problem, because each processor’s clock is different, trying to synchronize the clocks of the nodes is an even more difficult task. As
a result, there is a need for a synchronization mechanism that will synchronize the nodes with each other throughout the entire network.

When synchronizing system clocks, some of the clocks will have to be brought backwards to reflect the synchronised time. Note that the scope of this thesis only explains how the Holonic Logistics System provides the mechanism to determine the synchronised time, but not how the Holonic Logistics system will adapt to the new time.

An atomic clock is the most accurate type of timepiece in the world as it is designed to measure time using vibrations within atoms. Several atomic clocks are located in a number of places around the world and are used to establish the Coordinated Universal Time [38]. The Holonic Logistics System utilizes these atomic clocks to synchronize the clocks within each holons. Listing 3.36 shows the code that synchronizes the CPU clock with the atomic clock.

Synchronizing the holons’ internal clocks with the atomic clocks indirectly synchronizes the individual holons with one another. Because each clock is synchronized with the same atomic clock, it means that the holons will not differ very much, thereby resulting in a synchronized network.
Listing 3.36 Code that synchronizes system with an atomic clock.

In the event that the atomic clocks are not accurate, the Holonic Logistics System has a backup synchronization mechanism incorporated in itself. This backup mechanism involves keeping a variable of a certain value that is initiated during the system initialization. As outgoing external message will have the value embedded within it, the recipient of the message will then compare the value with the value of its own variable. If the incoming value is more updated, then the recipient holon will update its own variable with the message’s value. However, if the message’s value is not as recent as
the recipient’s value, then the update is not performed. Figure 3.27 and Figure 3.28 summarize the two synchronization mechanisms.

Figure 3.27 Logic flow of synchronization using atomic clock.
Figure 3.28 Logic flow of synchronization using messages.
3.3.3 Intelligence Mechanisms

To be able to adapt to a changing environment, there is a need for the Holonic Logistics System to be constantly monitoring the holonic environment. To do this, a new Intelligence Agent (IA) was created and added to the HLS. As Figure 3.29 indicates, this new agent’s functionality is invoked when a message has been analyzed.

Presently, there are two mechanisms of analysis performed: an analysis of traffic flow within a specific period of time, and an analysis of the amount of information transferred for a specific agent and for a specific holonic application. The former analyzes whether the HLS is being overworked and depending on the situation, will either increase or
decrease the processing rate of the HLS. The latter determines whether it would be more
efficient for the network performance to transfer a holonic agent to another holon.

3.3.3.1 Analyzing Traffic Flow

When an analyzed incoming message is forwarded to the Intelligence Agent, five
calculations are updated:

- number of messages / 30 seconds (Category 1)
- number of messages / 60 seconds (Category 2)
- number of messages / 600 seconds (Category 3)
- number of messages / 1800 seconds (Category 4)
- number of messages / 3600 seconds (Category 5).

These calculations provide the Holonic Logistics System with an idea of how much work
is being done by the system. Currently, the limit is set to 1.00. If the ratio that is
calculated for any of the categories is above the limit, then messages that are placed into
the system will be delayed for a period of time. This time frame depends on the category
that is being violated, as shown by Table 3.1.
Table 3.1 Delay times based on category

<table>
<thead>
<tr>
<th>Category</th>
<th>Delay Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Since the Message Analyzer Agent is associated with the Intelligence Agent, there is a need to communicate information of the calculated results to the MAA. Instead of having the Message Analyzer Agent store instances of different responses such as delay, the Message Analyzer will have an internal queue of objects that store the responses. Figure 3.30 clarifies this internal queue. The top section of the diagram illustrates the internal architecture of the MAA that contains the internal queue of responses. The bottom section of the diagram illustrates the steps taken by the MAA to ensure that the desired responses are performed. Note that because this architecture-mechanism pairing is used, there is a need for the Holonic Logistics System to have a set of keywords and key phrases dedicated for identifying responses, meaning that when the IA places a response object into the MAA’s response queue, the MAA will understand what action needs to be performed based on that keyword. Examples of keywords and key phrases are also found in the diagram.
3.3.3.2 Analyzing Amount of Transferred Information

The mechanism within the Intelligence Agent that is responsible for performing this analysis will have two queues: one for storing the information transferred for holonic agents and another for storing the information transferred for holonic applications. Each analyzed message indicates the target agent and target application, as well as the holon that is sending the message. Because of this, the IA will be able to tabulate how much information is being sent to each destination and from whom the information is being sent. Currently, the amount of information is set to 50 Mb. If the amount of information exceeds that amount, then the violating agent or application will be transferred to the holon that the host holon is communicating with. Figure 3.31 clarifies this mechanism.
Node 0 is found to have received more than 50MB of data from Holon E and Holon A for the holonic application "Ping Network". The application is sent to those holons.

3.4 Summary

The functionality, interface and intelligence issues demonstrate the complexity of the Holonic Logistics System as each of the different types of issues revealed a need. Those needs were necessary for the Holonic Logistics System to be able to become a fully distributed intelligence platform and to be efficient in handling the potential workload generated by the holonic environment. The provided solutions were designed and implemented with the concept that the HLS would be incorporated into a network of at least 50 nodes. As a result, some of the solutions may have seemed pointless at the time of explanation since a more centralized approach and a more closely-correlated system may have been an easier and simpler solution. However, it is essential to reemphasize that a centralized system whose subcomponents are very interdependent would have resulted in reduced system performance and reduced system efficiency as the number of nodes increased. With a focus on scalability, distributed and loosely-connected solutions were instead chosen.
4 APPLICATION OF THE HLS

This chapter demonstrates the functionality of the different holonic agents. A scenario has been devised to illustrate how the Holonic Logistics System can be applied in the real world. Throughout the chapter, a step-by-step analysis of how the scenario is created will be discussed. Through these steps the functionality of the Holonic Logistics System will be shown.

4.1 Creation of a Scenario

The following subsections will involve the step-by-step creation of a scenario that utilizes the Holonic Logistics System.

4.1.1 Step 1: Booting Up the HLS

Figure 4.1 shows the logical flow of how the HLS is initiated, Listing 4.1 shows the code that initiates the HLS, and Figure 4.2 and Figure 4.3 show screen shots of the logic flow being performed.
Execute platform launcher.

Platform launcher boots up Holonic Logistics System.

Holonic Logistics System is initialized and ready.

Figure 4.1 Launching of Holonic Logistics System.

```java
1 public static void main(String[] args) {
  2 System.out.println("About to initiate Holonic Technology Platform...");
  4 // instantiate instance of platform launcher
  5 ExecuteHolonicTechnologyPlatform LaunchingPlatform = new
      ExecuteHolonicTechnologyPlatform();
  8 // add event listeners to platform launcher
  9 PackageExecutable_RunAddAgentCodeEventListener
     RunAddAgentCodeEventListener = new
     PackageExecutable_RunAddAgentCodeEventHandler();
 10 LaunchingPlatform.AddRunAddAgentCodeEventListener(RunAddAgentCodeEventListener);
 13 PackageExecutable_LaunchNewHTPEventListener
     LaunchNewHTPEventListener = new
     PackageExecutable_LaunchNewHTPEventHandler();
 15 LaunchingPlatform.AddLaunchNewHTPEventListener(LaunchNewHTPEventListener);
 18 // in the first running of the holonic system, HTPMaster is created and added to the queue.
 19 HolonicTechnologyPlatformMaster HTPMaster = (HolonicTechnologyPlatformMaster)LaunchingPlatform.
      GetListOfHolonicTechnologyPlatforms().getFirst();
 21 HTPMaster.RunHTP();
 23 // HolonicTechnologyPlatformMasterInterface HTPMaster =
      GetHolonicTechnologyPlatformMasterInterface();
} 
```

Listing 4.1 Code that launches Holonic Logistics System.
Figure 4.2 Screen shot of platform launching.

Figure 4.3 Screen shot of launched platform.
Figure 4.3 shows the main menu that the HLS displays to the user upon start up. The
menu consists of two sections: System Details and System Options. As shown in the
figure, the System Details section conveys information such as the time and IP-address
that the HLS is using. The Systems Options section displays the different holonic agents
that are available to the user. Depending on the holonic agent that the user would like to
use, independent menus that display the options for the desired holonic agent will be
shown. Figure 4.4 and Figure 4.5 show examples of the Administration Agent’s menu
and of the System Agent’s menu.

Figure 4.4 Screen shot of Administration Agent menu.
At any point during the running of the Holonic Logistics System, if the user would like to check the node’s virtual resources, the user can do so by accessing the Systems Agent. The screen shot of the Systems Agent reveals the different types resources that the HLS is able to retrieve.

The highlighted portion of Figure 4.6 shows the time-synchronization feature of the Holonic Logistics System. As mentioned in Section 3.3.2, there is a need to synchronize the nodes on a distributed network. The screen shot shows the use of the atomic clocks to perform the synchronization.
4.1.2 Step 2: Creating a Virtual Network

A virtual network must first be created before a user or a holonic application can begin using the Holonic Logistics System to communicate with other nodes. There are two parts to creating a virtual network. The first part involves the creation of the virtual network object in the HLS and the second part involves the creation of the virtual links within the newly-created virtual network (VNET). Because there are several topologies that the HLS is capable of configuring, the user or holonic application can enter the nodes that should exist on the network and the HLS will build the network.
If there is a configuration that is not part of the library of topologies, then the virtual network can be built by having the clients of the HLS specify the links that should exist on the network through a text-file. For the purpose of this demonstration, the network topology in Figure 4.7 will be built. Because this topology is not in the library of topologies, the configuration will be done manually. Figure 4.8 shows the menu of the Virtual Networks Manager Agent. Figure 4.9, 4.10 and 4.11 show the state of the system before the creation of the virtual network, the creation of the virtual network, and the state of the system after the creation of the virtual network. Listing 4.2 shows the code that performs the creation of a virtual network, and Figure 4.12 is a representation of the network after the creation of the virtual networks on all nodes. Figure 4.13, 4.14 and 4.15 show the state of the system before the creation of a virtual link, the creation of a virtual link, and the state of the system after the creation of a virtual link. To conclude this subsection, Figure 4.16 summarizes the events that occur in this step.
Figure 4.8 Screen shot of Virtual Networks Manager Agent menu.

Figure 4.9 Screen shot of system state prior to creation of VNET.
Figure 4.10 Screen shot of creation of VNET.

Figure 4.11 Screen shot of system state after creation of VNET.
public void CreateNewVirtualNetworkEventReceived(PackageVirtualNetworksManagerAgent_CreateNewVirtualNetworkEvent Event)
{
 /*
 * 1)create on holonic layer
 * 2)create on cmns layer
 */
 /**********/

 String VirtualNetworkNameToCreate = Event.GetVirtualNetworkNameToCreate();

 PackageVirtualNetworksManagerAgent.QueueOfVirtualNetworks
 TempQueueOfVirtualNetworks =...

 NewVirtualNetwork = new PackageVirtualNetworksManagerAgent.VirtualNetwork();

 NewVirtualNetwork.SetVirtualNetworkName(VirtualNetworkNameToCreate);

 /**********/

 boolean VirtualNetworkExists =
 TempQueueOfVirtualNetworks.VirtualNetworkExists(VirtualNetworkNameToCreate);

 if(!VirtualNetworkExists) {

 PackageVirtualNetworksManagerAgent.VirtualNetwork Dummy
 = TempQueueOfVirtualNetworks.ModifyQueue(1, NewVirtualNetwork);

 }

 Listing 4.2 Code creating VNET.
Figure 4.12 Representation of network after creation of VNETs.

Figure 4.13 Screen shot of system state prior to creation of virtual links.
Figure 4.14 Screen shot of creation of virtual links.

Figure 4.15 Screen shot of system state after creation of virtual links.
Figure 4.16 Summary of events in this step.
4.1.3 Step 3: Creating a Holonic Agent

In the next step of the demonstration, Node F will create a new holonic agent called Demonstration Agent. Figure 4.17 – Figure 4.19 show examples of the system state before the creation of the holonic agent, the creation of the holonic agent, and the system state after the creation of the holonic agent. In the figures displaying the change in system state after the addition of the holonic agent. Notice the change in the structure of the package explorer.

![Figure 4.17 Screen shot of directory prior to creation of agent.](image-url)
Figure 4.18 Screen shot of creation of agent.

Figure 4.19 Screen shot of directory after creation of the agent.
The onus is on the developer of the new agent to ensure that the host node, which the agent is being developed on, properly integrates the Holonic Logistics System with the newly-created agent. Some of the functionalities to consider are basic requirements such as listing of the holonic agent with the HLS and the proper set up of communications structures within the HLS. Since the agent-template code is already shown in Section 3.1.8.1, the code will not be shown again in this section.

4.1.4 Step 4: Creating a Holonic Application

After Node F has created a new holonic agent, the next step in building the scenario will be the creation of a new holonic application on Node C. The new holonic application will be called Demonstration Application. Figure 4.20 – Figure 4.22 show the state of the system before the creation of the holonic application, the creation of the holonic application, and the state of the system after the creation of the holonic application.
Figure 4.20 Screen shot of directory prior to creation of the holonic application.

Figure 4.21 Screen shot of creation of the holonic application.
Once the holonic application has been created, the application has to register with the Holonic Logistics System. Figure 4.23 shows the console for a holonic application that is created using the generic holonic application template. Note that in the console menu, ‘Option i’ allows the application to perform this registration process. Figure 4.24 shows the list of applications prior to the registration, while Figure 4.25 shows the list of applications after the registration. To conclude this section, Figure 4.26 shows the updated holonic environment after the registration.
Figure 4.23 Screen shot of the holonic application console.

Figure 4.24 Screen shot of applications list prior to application registration.
Figure 4.25 Screen shot of applications list after application registration.

Figure 4.26 Updated view of network.
4.1.5 Step 5: Saving and Restoring the HLS

The step simulates the occurrence of a node first dropping out of the holonic environment and then rejoining the holonic environment, as shown in Figure 4.27.

![Diagram of failing nodes in the holonic environment](image)

Figure 4.27 Failing of nodes in the holonic environment.

When Node D rejoins the holonic environment, all the holonic node’s settings and states prior to the node failing will not be in the system since a fresh reboot of the HLS has been performed. This subsection shows that the step of saving the system’s information will actually combat the issue of lost settings and states. By saving the system’s information, upon rebooting the HLS, the system is able to restore previous settings. The saving can be done manually by the client of the HLS or it can be done automatically by the HLS itself. Regardless of the method of saving, the system’s information will be saved so that it can be restored later if needed. For the purpose of this demonstration, the manual method of saving will be used. Figure 4.28 shows how the system is saved, Figure 4.29 shows how the system is restored and Figure 4.30 shows the information being restored.
Figure 4.28 Screen shot of backing up system.

Figure 4.29 Screen shot of restoring system.
Because the information was saved, when the node re-enters the holonic environment, the node can restore the information. The screen shot in Figure 4.30 demonstrates this by exhibiting that “Demonstration Network” does exist after the information is restored, and that the virtual links within the network were also properly restored.

4.1.6 Step 6: Transferring Holonic Agents

In adding to the complexity of the scenario, the purpose of this step is to ensure that the newly-created Demonstration Agent is transferred to every other node within the holonic environment, as shown in Figure 4.31. Transferring a new holonic agent to other nodes
demonstrates how a holonic agents can be transferred from node to node without any issues.

Figure 4.31 Desired network environment.

As mentioned in Section 3.1.6, a comparison of software listings is made during the software transfer of holonic agents. The following diagrams, Figure 4.32 – Figure 4.33, illustrate a series of screen shots that display the software listing before and after a transfer in addition to the console displays. Figures 4.34 – 4.36 show the screen shots of the actual transfer. Note that to emphasize the difference, the screen shots have been modified to show only the holonic agent being transferred.
Figure 4.32 Screen shot of software listing prior to software transfer.

Figure 4.33 Screen shot of software listing after software transfer.
Figure 4.34 Screen shot of start of software transfer.

Figure 4.35 Screen shot of end of software transfer.
4.1.7 Step 7: Transferring Holonic Applications

The events in this step demonstrate the ability of the Holonic Logistics System to transport holonic applications throughout the network. Figure 4.37 shows the updated outlook of the network once the transfers have been completed. Figure 4.38 to Figure 4.40 show the screen shots of two nodes interacting to transfer the applications and the screen shot of the system state after the transfer. When looking at the state of the system after the transfer of the holonic application, it is important to note that since class files are being transferred, and not the Java source files, there will not be a directory created to hold the source files. Instead, an unique directory for the holonic application will be created to store the class-files. This distinction is important because there is a directory for the sources files and for the class files.
Figure 4.37 Desired updated network.

Figure 4.38 Screen shot of start of holonic application transfer.
Figure 4.39 Screen shot of directory after transfer of holonic application.

Figure 4.40 Screen shot of console of transferred holonic application.
4.1.8 Step 8: Updating Holonic Agent Software

Once a holonic agent has been created, there may be changes that are made to the agent package. These changes may be in the form of newer versions of existing classes or may be in the form of new classes within the package. Figure 4.41 shows the changes in the package on Node B in comparison to the package on Node C. Figure 4.42 to Figure 4.50 show the screen shots of software being updated for the holonic agent.

Figure 4.41 Changes in software listing.
Figure 4.42 Screen shot of start of software update.

Figure 4.43 Screen shot of software list being transferred during update.
Figure 4.44 Screen shot of information being transferred during update.

Figure 4.45 Screen shot of software list being sent to other node.
Figure 4.46 Screen shot of successful transfer.

Figure 4.47 Screen shot of end of software transfer.
Figure 4.49 Screen shot of software listing prior to software update.

Figure 4.48 Screen shot of other node ending software transfer.
4.1.9 Step 9: Agents Communicating within the Holonic Environment

This subsection demonstrates a simple scenario of a handshake process between two agents on two different nodes. The test code for this scenario is shown in Listing 4.3 and Listing 4.4, while the visual summarization is shown in Figure 4.51. Figure 4.52 and Figure 4.53 are screen shots of the scenario after the messaging has taken place.
1 PackageExecutable_AgentContainer CurrentAgentContainer =
2     null;
3 for (int i = 0; i < ListOfAgents.size(); i++)
4 {
5     CurrentAgentContainer =
6         PackageExecutable_AgentContainer) ListOfAgents.get(i);
7     if (CurrentAgentContainer.GetAgentName().
8             equals("DemonstrationAgent"))
9     {
10        //agent has been found so invoke method to handle
11        //external
12        //messages
13        try
14        {
15        //retrieve object
16        Object ExpectedAgentObject =
17        CurrentAgentContainer.GetAgentObject();
18        //identify which method to call
19        Class[] MethodParametersList = new Class[
20        {String.class, String.class,
21        String.class, String.class};
22        Method MethodToInvoke =
23        ExpectedAgentObject.getClass().getMethod("Se
24        ndExternalMessageToAgentOnAnotherNode",
25        MethodParametersList);
26        //set up input parameter
27        String[] DummyArgs =
28        new String[] {"Demonstration Network", "192.168.0.102",
29        "DemonstrationAgent", "Hello my name is
30        MasterColin");
31        try
32        {
33        //perform invocation
34        MethodToInvoke.invoke(ExpectedAgentObject,
35        (Object[]) DummyArgs);
36        }
37        catch (IllegalArgumentException e)
38        {
39        e.printStackTrace();
40        }
41        catch (IllegalAccessException e)
42        {
43        e.printStackTrace();
44        }
45        catch (InvocationTargetException e)
46        {
47        e.printStackTrace();
48        }
49    }
50   }
51   }
52   try
53   {
54   //perform invocation
55   MethodToInvoke.invoke(ExpectedAgentObject,
56   (Object[]) DummyArgs);
57   }
58   catch (IllegalArgumentException e)
59   {
60   e.printStackTrace();
61   }
62   catch (IllegalAccessException e)
63   {
64   e.printStackTrace();
65   }
66   catch (InvocationTargetException e)
67   {
68   e.printStackTrace();
69   }
70   }
71   }
72   try
73   {
74   //perform invocation
75   MethodToInvoke.invoke(ExpectedAgentObject,
76   (Object[]) DummyArgs);
77   }
78   catch (IllegalArgumentException e)
79   {
80   e.printStackTrace();
81   }
82   catch (IllegalAccessException e)
83   {
84   e.printStackTrace();
85   }
86   catch (InvocationTargetException e)
87   {
88   e.printStackTrace();
89   }
90   }
91   catch (IllegalArgumentException e)
92   {
93   e.printStackTrace();
94   }
95   catch (IllegalAccessException e)
96   {
97   e.printStackTrace();
98   }
99   catch (InvocationTargetException e)
100  {
101     //agent has been found so invoke method to handle
102     //external
103     //messages
104     try
105     {
106     //retrieve object
107     Object ExpectedAgentObject =
108     CurrentAgentContainer.GetAgentObject();
109     //identify which method to call
110    Class[] MethodParametersList = new Class[
111    {String.class, String.class,
112    String.class, String.class};
113    Method MethodToInvoke =
114    ExpectedAgentObject.getClass().getMethod("Se
115    ndExternalMessageToAgentOnAnotherNode",
116    MethodParametersList);
117    //set up input parameter
118    String[] DummyArgs =
119    new String[] {"Demonstration Network", "192.168.0.102",
120    "DemonstrationAgent", "Hello my name is
121    MasterColin");
122    try
123    {
124    //perform invocation
125    MethodToInvoke.invoke(ExpectedAgentObject,
126    (Object[]) DummyArgs);
127    }
128    catch (IllegalArgumentException e)
129    {
130    e.printStackTrace();
131    }
132    catch (IllegalAccessException e)
133    {
134    e.printStackTrace();
135    }
136    catch (InvocationTargetException e)
137    {
138    e.printStackTrace();
139    }
140    }
141    }
142    try
143    {
144    //perform invocation
145    MethodToInvoke.invoke(ExpectedAgentObject,
146    (Object[]) DummyArgs);
147    }
148    catch (IllegalArgumentException e)
149    {
150    e.printStackTrace();
151    }
152    catch (IllegalAccessException e)
153    {
154    e.printStackTrace();
155    }
156    catch (InvocationTargetException e)
157    {
158    e.printStackTrace();
159    }
160    }
161    }
162    try
163    {
164    //perform invocation
165    MethodToInvoke.invoke(ExpectedAgentObject,
166    (Object[]) DummyArgs);
167    }
168    catch (IllegalArgumentException e)
169    {
170    e.printStackTrace();
171    }
172    catch (IllegalAccessException e)
173    {
174    e.printStackTrace();
175    }
176    catch (InvocationTargetException e)
177    {
178    e.printStackTrace();
179    }
180    }
181    catch (IllegalArgumentException e)
182    {
183    e.printStackTrace();
184    }
185    catch (IllegalAccessException e)
186    {
187    e.printStackTrace();
188    }
189    catch (InvocationTargetException e)
190    {
191    e.printStackTrace();
192    }
193    }
194    catch (IllegalArgumentException e)
195    {
196    e.printStackTrace();
197    }
198    catch (IllegalAccessException e)
199    {
200    e.printStackTrace();
201    }
202    catch (InvocationTargetException e)
203    {
204    e.printStackTrace();
205    }
206    }
207    catch (IllegalArgumentException e)
208    {
209    e.printStackTrace();
210    }
211    catch (IllegalAccessException e)
212    {
213    e.printStackTrace();
214    }
215    catch (InvocationTargetException e)
216    {
217    e.printStackTrace();
218    }
219    }
220    catch (IllegalArgumentException e)
221    {
222    e.printStackTrace();
223    }
224    catch (IllegalAccessException e)
225    {
226    e.printStackTrace();
227    }
228    catch (InvocationTargetException e)
229    {
230    e.printStackTrace();
231    }
232    }
233    catch (IllegalArgumentException e)
234    {
235    e.printStackTrace();
236    }
237    catch (IllegalAccessException e)
238    {
239    e.printStackTrace();
240    }
241    catch (InvocationTargetException e)
242    {
243    e.printStackTrace();
244    }
245    }
Listing 4.4 Part 2 of test code.

Figure 4.51 Visual representation of messaging.
Figure 4.52 Screen shot of first message.

Figure 4.53 Screen shot of reply message.
4.1.10 Step 10: Intelligent Analysis

This final step demonstrates the intelligence possessed by the Holonic Logistics System. The intelligence exists in the form that the HLS is able to determine when there is a need to adapt to the network conditions. In this particular example, when an instance of Colin Agent on Node B has been excessively communicating with Node A, where an instance of Colin Agent does not exist, the HLS determines that there is a need to transfer Colin Agent from Node B to Node A as shown in the visual representation in Figure 4.54. This decision is made via the running of an intelligence scheme called PackageIntelligence-Agent_AnalyzeAgentCommunicationForSizeScheme.

Figure 4.54 Summary of events to transpire.
The functionality exists within the Holonic Logistics Agent, meaning that the intelligence operates in the background. As a result, the screen shots will show the step-by-step logic flow with the aid of the debugger. The purpose of the screen shots are to show that the steps taken are correct. Figure 4.55 to Figure 4.62 show the screen shots, where the first screen shot starts from the first calling of the intelligence analysis upon the receiving of a message to Colin Agent on Node B. Listing 4.5 to Listing 4.7 show the code implementation.

Figure 4.55 Screen shot of step-through of intelligence analysis 1.
Figure 4.56 Screen shot of step-through of intelligence analysis 2.

Figure 4.57 Screen shot of step-through of intelligence analysis 3.
Figure 4.58 Activity screen shot of step-through of intelligence analysis 4.

Figure 4.59 Activity screen shot of step-through of intelligence analysis 5.
Figure 4.60 Screen shot of step-through of intelligence analysis 6.

Figure 4.61 Screen shot of decision to transfer agent.
Figure 4.62 Screen shot of details of agent to be transferred.
String InputIPOfSource,
String InputAgentName,
String InputVirtualNetworkToAccess,
String InputMessageContent)
{
String ExtractedSourceField = InputIPOfSource;
String AgentBeingAccessed = InputAgentName;

//perform analysis
boolean TrackerForAgentExists = false;

PackageIntelligenceAgent_AgentCommunicationTracker
CurrentAgentCommunicationTracker = null;
for(int i = 0; i < ListOfHolonicAgents.size(); i++)
{
CurrentAgentCommunicationTracker =
(PackageIntelligenceAgent_AgentCommunicationTracker)ListOfHolonicAgents.get(i);
if((CurrentAgentCommunicationTracker.GetName()).equals(AgentBeingAccessed))
&& (CurrentAgentCommunicationTracker.GetNodeCommunicatingWith().equals(ExtractedSourceField)))
{
byte[] ByteArrayExtractedForMessageContent = InputMessageContent.getBytes();
int NumberOfBytesInArray = ByteArrayExtractedForMessageContent.length;
CurrentAgentCommunicationTracker.AddToNumberOfBytes(NumberOfBytesInArray);

//default set to 50 accesses from one source
if(CurrentAgentCommunicationTracker.GetNumberOfBytes() > 5000000)
{
//determining destination ip number
String DestinationIPNumber = ExtractedSourceField;

//determining virtual network to use
String VirtualNetworkToAccess = InputVirtualNetworkToAccess;

//determining agent package name
String PackageBeingAccessed = "Package_" + InputAgentName;

//checking to see if agent exists
String TypeOfAgent = "";

Listing 4.5 Part 1 of scheme used to decide agent transfer.
Listing 4.6 Part 2 of scheme used to decide agent transfer.
DestinationIPNumber,
83 VirtualNetworkToAccess);
84 }
85 }
86
87 TrackerForAgentExists = true;
88 }
89 }
90
91 //if tracker doesn't exist, add it and increment its
counter
92 if(!TrackerForAgentExists)
93 {
94
95
96 } PackageIntelligenceAgent_AgentCommunicationTracker
97 NewTrackerToAdd
98 = new PackageIntelligenceAgent_AgentCommunicationTracker
99 (AgentBeingAccessed, ExtractedSourceField, 0, 0);
100 byte[] ByteArrayExtractedForMessageContent
101 = InputMessageContent.getBytes();
102 int NumberOfBytesInArray =
103 ByteArrayExtractedForMessageContent.length;
104 NewTrackerToAdd.AddToNumberOfBytes(NumberOfBytes
105 InArray);
106
107 //since this scheme does not add variable to
108 intelligence list of variables, return null
109 return null;
110 }
4.2 Summary

In this chapter, the complexity of a simple holonic environment has been increased in incremental steps. With each step, an example has been provided to show how the Holonic Logistics System allows the collaboration amongst different holons to complete global and local goals. Through the demonstrated set of functionality, the HLS reveals that the system objectives and implementation objectives discussed in Chapter 2 have been satisfied, and that the implementation designs discussed in both Chapter 2 and 3 have been successfully implemented.
5 CONCLUSION

This chapter summarizes the research that has been accomplished to date on the Holonic Logistics System portion of the Holonic Technology Platform. First, the system objectives and implementation objectives of the Holonic Technology Platform will be discussed. In this discussion, there will be a review of the problems posed by present-day distributed intelligence systems, and a review of how the Holonic Technology Platform addresses some of those problems through the Holonic Logistics System. Next, a synopsis of the HLS infrastructure, benefits, and limitations will be deliberated. Finally, the chapter will conclude with an outline of possible future improvements that could be made to the HLS and with some final thoughts on the work accomplished thus far.

5.1 Review

A distributed system (DS) provides a cohesive network where each node’s hardware, software, and human users are integrated to create an entity that is capable of collaboration, cooperation and coordination [22].
5.1.1 Present-Day Problems

Two examples of present-day distributed systems that were examined were BitTorrent and Gnutella. Since these two systems were explained in detail in Section 1.5, a summary of the two systems will not be provided. However, problems posed by both systems will be discussed.

In Section 1.5.2, an analysis of both systems led to the discovery of several problematic issues. Those issues were:

- A need for the system to allow for more cooperation and collaboration amongst the network nodes. Nodes on the network were not fully utilizing the ability of all available nodes to perform system tasks.

- A need for more intelligence. When transferring files from seeders to leechers, BitTorrent sends the same blocks to each seeder. This does not fully utilize the distributed capability of a distributed system. For instance, a better way to transfer the files would be to have the intelligence to send different blocks to different seeders such as even-numbered blocks to one side of the network and odd-numbered blocks to the other side of the network. Then, both sides can transfer blocks between themselves. Using this way of transfer would increase the efficiency of leechers becoming seeders.
• A need for the system to reduce generated traffic. At times, every node on the network was flooded with a query request, even when the query request was not related to the recipient node. This flooding led to unnecessary generated traffic, which led to network problems such as the congestion of the network and the fragmentation of the network (shown in Figure 5.1).

![Fragmentation of network](image)

Figure 5.1 Fragmentation of network.

• A need for the system to have better security. Multiple applications may be using a DIS at any given time. However, messages destined for one of these
applications is made available to all the applications for analyzing. Furthermore, because there was no layer of security, the identity and the actions of the user were not masked.

- A need for the system to have better caching capability. Storing information in a distributed network is important. Without the ability for peers to interpret what each other has, the system will not be able to properly distributed information.

- A need for the system to have better routing mechanisms. Distributed networks provide an infrastructure that encourages efficient routing. By efficiently routing messages, the system will be able to find the best way to deliver a message based on metrics such as traffic rate and link stability. These metrics, however, will change depending on the conditions of the network. As a result, there is a need for those routing mechanisms to adapt to those changing metrics.

5.1.2 The Solution – Holonic Logistics System

The two examples of distributed intelligence systems discussed were built based on the multi-agent technology. Such systems are a collection of nodes, also called agents, which perform a set of tasks or satisfy a set of goals. In collaborating with one another to perform the tasks and to satisfy the goals, the agents communicate with each other via a specified communications protocol that is designed and incorporated into the architecture.
The Holonic Technology Platform is a distributed intelligence framework that is based on a hybrid model of the multi-agent technology. The ability of the HTP to make intelligent decisions and to adapt to a changing network is what defines it as a distributed intelligence system. Developed in Java because of Java’s ability to allow software components to be packetized and agentized, the Holonic Technology Platform utilizes both holons and agents to perform and satisfy both local and global tasks and goals. A holon is an agent entity that has the ability to cooperate with other holons in trying to achieve a global network goal(s) in addition to achieving its own local goal(s).

The Holonic Technology Platform has been partitioned into two layers to allow for faster development. Those two layers are the Holonic Logistics System and the Holonic Intelligence System. The former is responsible for the overall management of the platform, while the latter is responsible for making the appropriate decisions based on the current state of the network environment and the network resources. The Holonic Logistics System specifically addresses the previously-discussed shortcomings of present-day distributed intelligence systems.
5.1.2.1 HLS System Objectives

The system objectives of the Holonic Logistics System are:

1. To create an environment where each individual node on a network is able to cooperate with other individual network nodes in accomplishing a global task. In such an environment, the individual must also accomplish its own given task.

2. To allow each individual node on a network to have the ability to learn new responsibilities and new sets of functionality and to have the ability to adapt to a changing network environment.

These two system objectives provide a higher-level understanding of how this system will aid the Holonic Technology Platform in addressing those shortcomings. The commonality between all those shortcomings is that each requires some level of intelligent decision-making. However, intelligent decision-making is based on understanding the events that are occurring in the environment, where events denote activities occurring in not only the entire network, but also the activities occurring in each network participant. Hence, with the system objectives driving the design of the Holonic Logistics System, because nodes are aware of the network events and because nodes are capable of coordinating and collaborating activities with one another, the framework is able to have the necessary tools to perform the required intelligent decision-making.
5.1.2.2 HLS Implementation Objectives

The satisfied implementation objectives of the Holonic Logistics System that address the shortcomings specifically are:

1. A standard message structure must be used for all communications protocols. This is significant because it addresses the issue of a lack of security. The standard message structure guarantees that each HTP system that exists on an individual holon is capable of reading the core information such as the source and destination fields. Administration data and user data may be encrypted such that only the application with the proper decryption scheme is capable of understanding what the payload means. Figure 5.2 compares this solution to the problem.

Figure 5.2 Solution to privacy issue.
2. The system must be able to manage multiple virtual networks. This is significant because virtual networks provide a sense of purpose. For instance, a virtual network may consist of nodes that collaborate to accomplish Task A, which another virtual network may consist of nodes that collaborate to accomplish Task B. As a result, nodes on the same virtual network, to a certain extent, share some commonality. The benefit of having the ability to support virtual networks addresses the issue of an overly-congested network. Nodes within the same virtual network need only communicate with one another if there is no other need to communicate with nodes outside the network. By communicating with nodes within the same network instead of every node on the network, the HTP does not place such a heavy burden on the network. Figure 5.3 compares this solution to the problem.

3. Holons must be able to dynamically acquire intelligence. This addresses the issue of the need for the system to have better caching capabilities and routing mechanisms. First, intelligence in the caching capabilities context refers to information existing on network holons. Since information can be retrieved for all holons, and every holon is capable of identifying what information is on every other holon. Second, intelligence in the routing mechanisms context refers to the information regarding the holonic environment. A holon has the ability to retrieve the environment’s status either from another holon or from its own database. With the ability to understand how the environment is changing, the holon will then be able to adjust its activities accordingly.
Problem:
Query request traverses the entire network.

Solution:
By separating networks into virtual networks, only nodes that are collaborating with one another need to communicate with one another. As a result, network traffic is reduced.

Figure 5.3 Solution to generated traffic issue.

5.2 Synopsis

5.2.1 HLS Infrastructure

Every node using the Holonic Logistics System is considered a holon. This means that nodes using the holonic environment are able to collaborate with one another by
coordinating activities occurring on the individual nodes. This ability is facilitated by the holon’s architecture, which is composed of agents. Figure 5.4 demonstrates this relationship. Each agent has a set of tasks to perform and do not work with other agents in completing those tasks.

![Figure 5.4 Relationship between holon and agents.](image)

Because of such a loose cohesion between the agents, agents can be added, removed, and updated without interfering with other agents. An important point to note is that when agents are removed, the set of functionality offered by those removed agents is also removed. To prevent this sudden loss of core functionality, such as the ability to send external messages to other holons, the HLS has a set of core agents that should always be in the system. These set of agents are listed in Table 5.1.
<table>
<thead>
<tr>
<th>Agent Name</th>
<th>Agent Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administration Agent</td>
<td>Monitors and controls system administration functionality such as system shutdown, system update and system initialization.</td>
</tr>
<tr>
<td>Algorithms Agent</td>
<td>Initiates algorithms and manages settings of algorithms within the Holonic Logistic System</td>
</tr>
<tr>
<td>Applications Manager Agent</td>
<td>Monitors and controls all functionality related to an application registered with the system.</td>
</tr>
<tr>
<td>Console Agent</td>
<td>Displays system outputs to the user and accepts system inputs from the user.</td>
</tr>
<tr>
<td>Database Agent</td>
<td>Backs up the database of each agent before system shutdown, and restores database of each agent during system initialization.</td>
</tr>
<tr>
<td>Development Agent</td>
<td>Creates new agents and holonic applications</td>
</tr>
<tr>
<td>Intelligence Agent</td>
<td>Contains analytical schemes that analyze different network aspects of the system. Some of the areas analyzed include network traffic and network resources.</td>
</tr>
<tr>
<td>Message Analyzer Agent</td>
<td>Analyzes all incoming and outgoing messages for internal and external purposes.</td>
</tr>
<tr>
<td>Messenger Agent</td>
<td>Sorts all incoming and outgoing messages for internal and external purposes.</td>
</tr>
<tr>
<td>Software Transfer Agent</td>
<td>Checks to ensure that each node is updated with the latest software version of the Holonic Logistic System</td>
</tr>
<tr>
<td>System Agent</td>
<td>Monitors operating system attributes such as number of threads, number of running applications, etc.</td>
</tr>
<tr>
<td>Version Tracking Agent</td>
<td>Keeps track of software versions within the Holonic Logistic System</td>
</tr>
<tr>
<td>Virtual Networks Manager Agent</td>
<td>Monitors and controls all functionality related to a virtual network.</td>
</tr>
</tbody>
</table>
### 5.2.2 HLS Benefits

As inferred in Section 5.2.1, one of the most significant benefits offered by the Holonic Logistics System’s infrastructure is its ability to add, remove, and update agents without interfering with the functionality of other agents. Because this benefit exists, the following spin-off benefits are offered by the Holonic Logistics System:

1. **Ability to develop an agent based on a template provided by the HLS.** The system will create an agent with a core set of functionality. Developers have the ability to add to the functionality. The key is that the core functionality must remain in the system in order for the agent to function in the holonic environment.

2. **Ability to develop an application based on a multi-agent architecture.** The HLS creates an application that is composed of agents. These agents are necessary for the application to interface with the Holonic Logistics System, and provide a core set of functionality that allows the application to use the HLS.

3. **Ability to transfer an agent from one holon to another without interfering with agent functionality that is not core.** In the event that a new software version has been released for a core agent, the updated software can still be transferred to a holon. As long as the core agent is not removed from the system, the updated software will be loaded into the HLS.
4. Ability to transfer an application from one holon to another. The agents that form the architecture of the application will be transferred as a package so that the recipient holon will have the complete application.

5. Ability to add algorithms and intelligence schemes to the system. Algorithms belonging to the Algorithms Agent enhance the Holonic Logistics System with the ability to perform more functionality. An algorithm that exemplifies such functionality would be an algorithm that coordinated activities amongst the holons. The algorithm may focus on how to decide event synchronization amongst the systems. The Intelligence Agent, on the other hand, enhances the Holonic Logistics System with the ability to adapt to the changing holonic environment. For instance, some schemes may focus on analyzing the local traffic generated by the host system and on the available resources offered by the host system, while other schemes may focus on analyzing the global traffic generated by the holonic system and the available resources offered by the system. Regardless of the focus, the intelligence schemes aid the holons in adapting to the environment by triggering events to accommodate those changes.

In addition to benefits offered by the Holonic Logistic System’s infrastructure, some other features of the HLS are:

1. Ability to form groups of holons. Holons that share a common goal or functionality are able to form their own groups so that collaborative activities related to this common goal will be restricted to within the group.
2. Ability to monitor the available local hardware and virtual resources.

3. Ability to efficiently utilize threads. The Holonic Logistics System has three threads that are constantly running. These threads support the communications functionality and the console functionality. Each functionality that is invoked by the system will trigger the creation of one thread. After the functionality has been satisfied, the thread will terminate itself. This method of thread usage guarantees that thread resources will only be utilized necessary.

4. Ability to back up information. Although the Holonic Logistics System is a distributed system, there is still a lot of information that needs to be stored. The HLS provides a backup mechanism that allows that information to be backed up in the event of a system failure. Furthermore, in the event of a system failure, if a backup was previously made, the backup can be used to update the system to a state prior to the system failure.

Individually, the previously-mentioned benefits offer features that clients and users of the system would find to be very useful tools and functionality. However, as a collective set of benefits, the Holonic Logistics System is empowered with the ability to satisfy its two system objectives, as reviewed in Section 5.1.2.1. By satisfying those two objectives, the Holonic Logistics System offers a distributed intelligence system that is intelligent enough to be able to adapt to changing network environments, intelligent enough to learn how to adapt to changing network environments, and intelligent enough to coordinate
network activities by allowing holons to collaborate with one another in satisfying and completing both local and global goals and tasks.

5.2.3 HLS Limitations

5.2.3.1 Concurrency

Concurrency refers to the simultaneous running of two or more execution flows. In the context of the Holonic Logistics System, the limitations posed by concurrency issues refer to the parallel execution flows of tasks within the HLS. Presently, the three threads that are constantly running perform the following functions:

- Listen for messages from applications utilizing the Holonic Logistics System,
- Listen for messages from the Holonic Communications System,
- Display user console.

In theory, these three threads should share the processor fairly and should appear as if they were running the abovementioned tasks simultaneously. However, the Holonic Logistics System does not always function this way. There is often a lag in displaying the user outputs and the user console when the system is busy receiving messages from both the upper and lower layers. This lag is often increased when the system has to perform a series of events in response to the message. The cause of the lag is the blocking function call that is incorporated into the function that listens for messages.
When there are messages to be pulled from the holon's input port, the thread responsible for listening will not release the processor until it is finished receiving the messages.

In addition to these tasks, other functions that experience concurrency issues are the tasks performed by event threads. As discussed earlier, event threads are only created when an event is triggered. When there are a large number of event threads handling triggered events, the Holonic Logistics System also experiences a lag in performance. However, the cause of the lag produced by such events is not limited to blocking function calls. Other causes could be accessing other resources such as memory and accessing databases through the use of text files.

A possible solution to improving the concurrency issue involves two phases. The first phase involves limiting the blocking of the processor to a set time period. The second phase involves separating the threads into two types of categories – user threads and service threads.

The first phase could be implemented with the aid of timers. When a blocking function is using the processor, a timer is started. When the timer has timed out, the thread has to release the processor and wait until its next turn to execute the task once again. A problem that could arise from this implementation is that in the case of the HLS, messages may be dropped if the thread does not receive those messages fast enough. Hence, with this implementation, the receiving of the messages from the input port must be performed quickly and efficiently.
The second phase could be implemented by having an agent that monitors the thread usage. Through this design, user threads employ a cooperative strategy whereas service threads employ a pre-emptive strategy [28]. A cooperative strategy does not suspend any of the threads from using the processor. Instead, user threads are only allowed to utilize the processor when it is available. A pre-emptive strategy suspends threads that are using the processor, so that a service thread may utilize the processor. Having these two categories of threads would mean that threads of higher importance will, on the condition that the threads utilizing and waiting for the processor are of lower priority, have access to the processor when they require it.

5.2.3.2 Port Forwarding

When several computers are connected to a router, the router assigns localized IP-numbers to those computers. The reason for the IP-assignments is that the computers have to share the one IP-number that is connected from the external world to the router. In such a situation, if the HLS is installed on every node, a problem arises since the HLS uses one designated port. That problem is illustrated in Figure 5.5.
This problem can be alleviated in two ways – a hardware solution and a software solution. The first hardware solution is simple – guarantee that each system exists exclusively on only one intranet when the system has to communicate with systems outside of the intranet. Since each holon is connected to one gateway, this problem will be alleviated.

The software solution involves the HLS having the functionality to determine if the port is in use by another application or by another Holonic Logistics System. If either case is
true, then the HLS will attempt to list another port number with the router. The alternative port number, however, will have to exist between a set range of port numbers. In order to be able to accommodate the range of port numbers, the HLS will have to send the message to every port number in that range. Although this solution does work, there is a problem that could arise from this implementation. The traffic generated by the redundant messages to the ports could be significant enough to congest the network. As a result, there is a need to perform some testing to see the effectiveness of the solution.

5.2.3.3 Communications Infrastructure

Although this limitation is out of the control of the Holonic Logistics System, it is still a factor that needs to be considered. The amount of data that is sent throughout the holonic environment and the rate at which the data is sent throughout the holonic environment depends heavily on the capability of the communications infrastructure that is being used to perform the actual communications such as the sending and routing of the messages. If the communications infrastructure takes too long to route the messages, then the response of the HLS on the holons will be slow since the messages are arriving at the destination holons slowly. Furthermore, if the communications infrastructure cannot handle the amount of information that is being sent throughout the holonic environment, then there may be lost packets during the exchange of information. In such a case, the HLS on the message sender will have to resend the message, thus wasting more time and resources.
5.3 Future Improvements

In addition to implementing the solutions discussed in Section 5.2, there are several features that should be implemented into the next version of the Holonic Logistics System.

5.3.1 User Console Loading

Currently, the user console for each agent is stored in the Console Agent. This means that when an agent has a menu that needs to be displayed as an output to the user, that menu has to be hardcoded into the Console Agent. This is not ideal because it indicates that when an agent is transferred from one holon to another, the code that contains the menu in the Console Agent will also have to be transferred. This is problematic because a software update will have to be performed on the Console Agent, which is unnecessary work.

A better alternative would be to hardcode the menu into the agent file. The menu would be placed inside a function called PopulateQueueOfMenuOptions(). The benefit of placing the menu code into the agent file is that when transferring an agent, there is no need to perform a software update on the Console Agent in order for the newly-transferred agent to be able to output its menu to users on the new holon. The menu loading mechanism is simple. Upon startup, the agent will be asked to send the menu to the Console Agent. The PopulateQueueOfMenuOptions() function will then be invoked, as shown in Listing 5.1, so that it sends the menu to the Console Agent, where a copy of
the menu will be kept. The benefit of such a mechanism is that the menu will only need to be loaded once, and it will be done at the start of the system.

```java
public void PopulateQueueOfMenuOptions()
{
    /*
    * Sample Menu:
    *
    * a) Add animal //takes in animal name and animal type
    */
    String OptionValueA = "A";
    String OptionTitleA = "Add animal";
    PackageAbstractComponents_AbstractMenuOption MenuOptionA =
    new PackageAbstractComponents_AbstractMenuOption(OptionValueA,
      OptionTitleA);
    String TypeOfInputForOptionAParameter1 = "String";
    String RequestMessageForOptionAParameter1 = "Please enter animal name";
    PackageAbstractComponents_AbstractOptionInputParameter OptionAParameter1 =
    new PackageAbstractComponents_AbstractOptionInputParameter(.TypeOfInputFor
      OptionAParameter1,
      RequestMessageForOptionAParameter1);
    MenuOptionA.GetQueueOfOptionInputParameters().add(OptionAParameter2);
    //add menu option to agent's input queue of menu options
    this.GetQueueOfMenuOptions().add(MenuOptionA);
}
```

Listing 5.1 Example of PopulateQueueOfMenuOptions method.

### 5.3.2 Internal Messaging System

There is a need for the internal messaging system to migrate away from the use of events to handle internal communications. The reason for this need is because each event
requires a separate piece of handling code. As a result of this requirement, the number of classes will increase with the number of events. Rather than having to add to the complexity of the system in terms of the number of classes, a better alternative would be to have one generic messaging system that would not require so many classes.

In this generic messaging system, there is one function that is called when a message has to be sent. This function is `HandleInternalEventReceived(PackageExecutable_SwitchMechanismHandleInternalMessagesEvent Event)` and is shown in Listing 5.2. The mechanism that handles the messaging system is as follows:

1. When an agent has to communicate with another agent on the same node, an event is triggered to call `HandleInternalEventReceived(...)`.  

2. `HandleInternalEventReceived(...)` checks to see which agent is being accessed, and sends the message to that agent.

3. Instead of having multiple classes to handle different cases of the messages, there is only one class that handles all the messages that are sent to an agent. That class is called `ServiceInternalMessages-EventHandler` and exists in every agent. When that class is accessed, the function `ServiceInternalMessagesEventReceived(...)` will be invoked to handle the newly-received internal message. A sample code sniplet is shown in Listing 5.3.
public synchronized void HandleInternalMessagesEventReceived(Package Executable_SwitchMechanismHandleInternalMessagesEvent Event) {
  PackageAbstractComponents_AbstractInternalMessage StoredInternalMessage = Event.GetInternalMessageBeingHandled();
  String AgentBeingAccessed = StoredInternalMessage.GetAgentBeingAccessed();
  if (AgentBeingAccessed.equals("AdministrationAgent")) {
    PackageAbstractComponents_AbstractInternalMessage DummyInternalMessage = Event.GetHolonicTechnologyPlatformMaster().GetAdministrationAgent().GetDatabaseMaster().ModifyQueue(1, StoredInternalMessage);
    Event.GetHolonicTechnologyPlatformMaster().FireServiceAdministrationInternalMessageQueueEvent();
  } else if (AgentBeingAccessed.equals("AlgorithmsAgent")) {
    PackageAbstractComponents_AbstractInternalMessage DummyInternalMessage = Event.GetHolonicTechnologyPlatformMaster().GetAlgorithmsAgent().GetDatabaseMaster().ModifyQueue(1, StoredInternalMessage);
    Event.GetHolonicTechnologyPlatformMaster().FireServiceAlgorithmsAgentInternalMessageQueueEvent();
  } else if (AgentBeingAccessed.equals("ApplicationsManagerAgent")) {
    PackageAbstractComponents_AbstractInternalMessage DummyInternalMessage = Event.GetHolonicTechnologyPlatformMaster().GetApplicationsManagerAgent().GetDatabaseMaster().ModifyQueue(1, StoredInternalMessage);
    Event.GetHolonicTechnologyPlatformMaster().FireServiceApplicationsManagerAgentInternalMessageQueueEvent();
  };

Listing 5.2 Example of HandleInternalEventReceived method.
public synchronized void ServiceInternalMessagesEventReceived(
    PackageColinAgent_ServiceInternalMessagesEvent Event)
{
    PackageAbstractComponents_AbstractInternalMessage
        DummyInternalMessage = null;

    PackageAbstractComponents_AbstractInternalMessage
        ExtractedInternalMessage =
        Event.GetHolonicTechnologyPlatformMaster().
            GetColinAgent().GetDatabaseMaster().ModifyQueue(0,
            DummyInternalMessage);

    String ExtractedFunctionBeingAccessed =
        ExtractedInternalMessage.GetFunctionBeingAccessed();
    String MessageContent =
        ExtractedInternalMessage.GetMessageContent();

    if (ExtractedFunctionBeingAccessed.equals("Run User
        Selected Option"))
    {
        String[] ResultingArrayFromSplit =
            new String [MessageContent.split("<Content>").length];
        ResultingArrayFromSplit = MessageContent.split("<Content>");

        // all information can be found in this array... do as
        // desired...
    }

    else if (ExtractedFunctionBeingAccessed.equals("..."))
    {
        /*
        * 1. whatever the function being accessed is, the
        * functionality can be placed inside the agent class
        * 2. this will allow easier transference as the agent
        *    class
        *    is the main focus of software transfer now
        */
    }
}

Listing 5.3 Example of ServiceInternalMessagesEventReceived method.
5.3.3 Agent Template

As the Holonic Logistics System increases in complexity, there will be more functionality that will be required of an agent. It is thus important to keep increasing the complexity of the agent. Some functionality that may be of use are:

- be able to auto-update of code with other holons,
- be able to auto-save information into database,
- be able to auto-remove resources that are not required,
- be able to gather information from all similar holons and agents.

5.3.4 Data Logging

Currently, text-files are being used to store information. However, this presents a problem. Reading and writing from text-files is incredibly slow and becomes worse as the size of the text-file increases. An alternative solution is to use channels. Channels, through the use of buffers, allow for the reading and writing of information by blocks, as opposed to reading and writing by bytes. This allows for more efficiency and for quicker read and write speeds.
5.4 Final Remarks

The Holonic Logistics Systems is a simple model of a distributed intelligence system that overcomes issues in present-day systems. More importantly, the research accomplished in this thesis provides an example of how distributed intelligence systems can be developed, and provides a framework for the development of other distributed intelligence systems. Although the concepts and definitions of a distributed intelligence systems are presently being debated by members of the intellectual committee, it is anticipated that this thesis will play a role in defining what distributed intelligence systems truly are.
APPENDIX A

1. Core Development Folder (~53,000 lines of code)
   a. Package Abstract Components (15 files)
   b. Package Administration Agent (15 files)
   c. Package Algorithms Agent (23 files)
   d. Package Algorithms (2 files)
   e. Package Applications Manager Agent (76 files)
   f. Package Console Agent (48 files)
   g. Package Database Agent (20 files)
   h. Package Development Agent (16 files)
   i. Package Executable (16 files)
   j. Package GUI Agent (5 files)
   k. Package Intelligence Agent (13 files)
   l. Package Message Analyzer Agent (9 files)
   m. Package Messenger Agent (59 files)
   n. Package Software Transfer Agent (31 files)
   o. Package Systems Agent (41 files)
   p. Package Test Functionality (34 files)
   q. Package Version Tracking Agent (7 files)
   r. Package Virtual Networks Manager Agent (62 files)
2. Proprietary Development Folder (~500 lines of code)
   a. Package Agents
      i. Package Colin Agent (5 files)
   b. Package Algorithms (1 files)

3. Applications Development Folder (~7,000 lines of code)
   a. Package Components Of Generic Application (100 files)
   b. Package Demonstration Application (100 files)
APPENDIX B

1 package Package_ExecutablePackage;
2
3 import java.io. * ;
4 import java.util. * ;
5 import DynaCode.DynaCode;
6
7 public class ExecuteHolonicTechnologyPlatform implements Serializable
8 {
9     /*************************/
10     /* variable declarations */
11     /*************************/
12     
13     private List RunAddAgentCodeEventListeners;
14     private List LaunchNewHTPEventListeners;
15     private boolean RunHolonicTechnologyPlatform;
16     private LinkedList ListOfHolonicTechnologyPlatforms;
17     private HolonicTechnologyPlatformMasterInterface HTPInterfaceInstance;
18     
19     20     /***************
21     /* constructor */
22     /***************
23     
24     25 public ExecuteHolonicTechnologyPlatform()
25     {
26         RunAddAgentCodeEventListeners = new ArrayList();
27         LaunchNewHTPEventListeners = new ArrayList();
28         RunHolonicTechnologyPlatform = true;
29         
30         ListOfHolonicTechnologyPlatforms = new LinkedList();
31         
32         HTPInterfaceInstance = null;
33         
34         //initial setup includes an htp
35         HolonicTechnologyPlatformMaster HTPMaster = new HolonicTechnologyPlatformMaster(this);
36         ListOfHolonicTechnologyPlatforms.add(HTPMaster);
37     }
38     
39     40     
41     42     /************
43     /* accessor */
44     /************
45     
46     47 public LinkedList GetListOfHolonicTechnologyPlatforms()
48 { 
49     return ListOfHolonicTechnologyPlatforms;
50 }
51
52 public HolonicTechnologyPlatformMasterInterface
53 GetHolonicTechnologyPlatformMasterInterface()
54 {
55     return HTPInterfaceInstance;
56 }
57
58
59 /********************************/
60 /* event-related code */
61 /********************************/
62 //RunAddAgentCodeEvent
63 public synchronized void
64 AddRunAddAgentCodeEventListener(PackageExecutable_RunAddAgentCodeEventListener Listener)
65 {
66     RunAddAgentCodeEventListener.add(Listener);
67 }
68
69 public synchronized void
70 RemoveRunAddAgentCodeEventListener(PackageExecutable_RunAddAgentCodeEventListener Listener)
71 {
72     RunAddAgentCodeEventListener.remove(Listener);
73 }
74
75 public synchronized void
76 FireRunAddAgentCodeEvent(HolonicTechnologyPlatformMaster InputHTPMaster)
77 {
78     // creates new event and fires it to be handled
79     PackageExecutable_RunAddAgentCodeEvent NewEvent =
80     new PackageExecutable_RunAddAgentCodeEvent(this, InputHTPMaster);
81     Iterator Listeners =
82         RunAddAgentCodeEventListener.iterator();
83
84     boolean test = Listeners.hasNext();
85
86     while(Listeners.hasNext())
87     {
88         ((PackageExecutable_RunAddAgentCodeEventListener)Listeners.next()).RunAddAgentCodeEventReceived(NewEvent);
89     }
90 }
91
92 //LaunchNewHTPEvent
93 public synchronized void
94 AddLaunchNewHTPEventListener(PackageExecutable_LaunchNewHTPEventListener Listener)
95 {
96     LaunchNewHTPEventListener.add(Listener);
97 }
98
99 public synchronized void
100 RemoveLaunchNewHTPEventListener(PackageExecutable_LaunchNewHTPEventListener Listener)
101 {
102     LaunchNewHTPEventListener.remove(Listener);
103 }
104
105 public synchronized void
106 FireLaunchNewHTPEvent(HolonicTechnologyPlatformMaster InputHTPMaster)
107 {
108     // creates new event and fires it to be handled
109     HolonicTechnologyPlatformMaster NewEvent =
110     new HolonicTechnologyPlatformMaster(this, InputHTPMaster);
111     Iterator Listeners =
112         LaunchNewHTPEventListener.iterator();
113
114     boolean test = Listeners.hasNext();
115
116     while(Listeners.hasNext())
117     {
118         ((PackageExecutable_LaunchNewHTPEventListener)Listeners.next()).LaunchNewHTPEventReceived(NewEvent);
119     }
120 }
Listener)

94 {  
95     LaunchNewHTPEventListeners.add(Listener);
96 }
97
98
99 public synchronized void
100 RemoveLaunchNewHTPEventEventListener(PackageExecutable_LaunchNewHTPEventListe
101 nListener)
102 
103 LaunchNewHTPEventListeners.remove(Listener);
104 } 
105
106 /*creates new event and fires it to be handled
107 PackageExecutable_LaunchNewHTPEvent NewEvent =
108     new PackageExecutable_LaunchNewHTPEvent(this,
109     this);
110 Iterator Listeners =
111     LaunchNewHTPEventListeners.iterator();
112 boolean test = Listeners.hasNext();
113 while(Listeners.hasNext())
114 {
115     ((PackageExecutable_LaunchNewHTPEventEventListener)Listeners.next()).LaunchNe
116 wHTPEventReceived(NewEvent);
117 }
118
119
120
121 *************
122 */ utility */
123 *************
124
125 public void InitiateHTP()
126 {
127     /*
128     * not implemented
129     */
130 }
131
132
133 public void ShutDownHTP()
134 {
135     HolonicTechnologyPlatformMaster HTPMaster
136     = (HolonicTechnologyPlatformMaster)this.GetListOfHolonicTechnology
137     Platforms().removeFirst();
138     //make sure that all references to htp have been removed
139     //
140     HTPMaster.DestroySelf();
141 
142 try
143 {
144 
145
HTPMaster.finalize();

} catch (Throwable e1) {
    e1.printStackTrace();
}

//call garbage collector
System.gc();
System.runFinalization();

HolonicTechnologyPlatformMaster HTPMaster = (HolonicTechnologyPlatformMaster) this.GetListOfHolonicTechnologyPlatforms().removeFirst();

//make sure that all references to htp have been removed
HTPMaster.DestroySelf();

try {
    HTPMaster.finalize();
}

} catch (Throwable e1) {
    e1.printStackTrace();
}

//call garbage collector
System.gc();
System.runFinalization();
System.exit(0);

private static HolonicTechnologyPlatformMasterInterface GetInstanceOfHolonicTechnologyPlatformMasterInterface() {
    String CurrentDirectory = System.getProperty("user.dir");
    DynaCode dynacode = new DynaCode();
    dynacode.addSourceDir(new File(CurrentDirectory + "\bin\Package_ExecutablePackage\"));
    return (HolonicTechnologyPlatformMasterInterface) dynacode.newProxyInstance(HolonicTechnologyPlatformMasterInterface.class, CurrentDirectory + "\bin\Package_ExecutablePackage.HolonicTechnologyPlatformMasterInterface" );

}
195 } 
196 
197 198 public void StartHTP() 
199 { 
200     HTPInterfaceInstance = GetInstanceOfHolonicTechnologyPlatformMasterInterface(); 
201     HTPInterfaceInstance.RunHTP(); 
202 } 
203 
204 /**********************************/
205 /* software transfer functionality */
206 /**********************************/
207 
208 public static void TestIfSoftwareHasBeenTransferredSuccessfully() 
209 { 
210     System.out.println("SYSTEM INFORMATION: The following 
211         class, ExecuteHolonicTechnologyPlatform, has been loaded successfully
212         after the transfer of byte codes."); 
213 } 
214 
215 /********/
216 /* main */
217 /********/
218 
219 public static void main(String[] args) 
220 { 
221     System.out.println("About to initiate Holonic Technology
222         Platform..."); 
223 
224     //instantiate instance of platform launcher
225     ExecuteHolonicTechnologyPlatform LaunchingPlatform = new
226         ExecuteHolonicTechnologyPlatform(); 
227 
228     //add event listeners to platform launcher
229     PackageExecutable_RunAddAgentCodeEventListener
230         RunAddAgentCodeEventListener = new
231         PackageExecutable_RunAddAgentCodeEventListener(); 
232     LaunchingPlatform.AddRunAddAgentCodeEventListener( 
233         RunAddAgentCodeEventListener); 
234 
235     PackageExecutable_LaunchNewHTPEventListener
236         LaunchNewHTPEventListener = new
237         PackageExecutable_LaunchNewHTPEventListener(); 
238     LaunchingPlatform.AddLaunchNewHTPEventListener( 
239         LaunchNewHTPEventListener); 
240 
241     //in the first running of the holonic system, HTPMaster is created and added to the queue.
242     HolonicTechnologyPlatformMaster HTPMaster
243         = (HolonicTechnologyPlatformMaster)LaunchingPlatform.
244             GetListOfHolonicTechnologyPlatforms().getFirst(); 
245     HTPMaster.RunHTP();
// HolonicTechnologyPlatformMasterInterface HTPMaster =
GetHolonicTechnologyPlatformMasterInterface();
// LaunchingPlatform.StartHTP();
}
APPENDIX C

1 package Package_ExecutablePackage;
2
3 /***************************************************************************/
4 /* Import Declarations */
5 /***************************************************************************/
6 import java.net.*;
7 import java.util.*;
8 import java.io.*;
9 import java.lang.Thread;
10 import java.lang.reflect.InvocationTargetException;
11 import java.lang.reflect.Method;
12 import java.lang.reflect.Method;
13 import Package_AbstractComponents.*;
14 import Package_ExecutablePackage.PackageExecutable_ConsoleReader;
15 import Package_GUIAgent.*;
16 import Package_IntelligenceAgent.*;
17 import Package_VirtualNetworksManagerAgent.*;
18 import Package_MessageAnalyzerAgent.*;
19 import Package_DatabaseAgent.*;
20 import Package_DevelopmentAgent.*;
21 import Package_ConsoleAgent.*;
22 import Package_AlgorithmsAgent.*;
23 import Package_ApplicationsManagerAgent.*;
24 import Package_AdministrationAgent.*;
25 import Package_MessengerAgent.*;
26 import Package_SoftwareTransferAgent.*;
27 import Package_SystemsAgent.*;
28 import Package_VersionTrackingAgent.*;
29 import Agents.Package_ColinAgent.*;
30 import Package_ExecutablePackage.;
31 /***************************************************************************/
32
33 public class HolonicTechnologyPlatformMaster implements Serializable
34 {
35 /***************************************************************************/
36 /* variable declarations */
37 /***************************************************************************/
38 private ExecuteHolonicTechnologyPlatform LaunchingPlatform;
39
40 private boolean Stop;
41 private boolean StopDisplayingConsoleMenu;
42 private boolean FirstTimeDisplayingConsoleMenu;
43}
private String ComputerName;
private long SystemLocalTime;
private long SystemTimeStamp;
private Calendar UpdatedCalendar;
private int NumberOfCoreAgents;
private String NumberOfCoreAgentsAsString;
private int NumberOfPropAgents;
private String NumberOfPropAgentsAsString;
private Timer AtomicSynchronizationTimer;

private InetAddress OwnIPAddress;
//this variable contains the ip-number AND the computer name
private String OwnIPNumber;
//this variable contains only the ip-number
private int IncomingCommunicationsPort;
//input port for cmns layer
private int OutgoingCommunicationsPort;
//output port for cmns layer
private int IncomingApplicationsPort;
//input port for apps layer
private int OutgoingApplicationsPort;
//output port for apps layer
private int IncomingHolonicPortOnApplicationsLayer;
private int CurrentMessageCountValue;
private DatagramSocket IncomingCommunicationsSocket;
//input socket for cmns layer
private DatagramSocket IncomingApplicationsSocket;
//input socket for apps layer
private DatagramSocket IncomingApplicationsSocket;

private PackageAbstractComponents_AbstractInputOutputOueue MainInputQueue;
private PackageAbstractComponents_AbstractInputOutputOueue MainOutputQueue;
private PackageAbstractComponents_AbstractInputOutputOueue BackupMainOutputQueue;
private LinkedList ListOfAgents;

private PackageConsoleAgent_ConsoleAgent ConsoleAgent;
//MUST ALWAYS BE FIRST AGENT TO BE CREATED
private PackageAdministrationAgent_AdministrationAgent AdministrationAgent;
private PackageAlgorithmsAgent_AlgorithmsAgent AlgorithmsAgent;
private PackageApplicationsManagerAgent_ApplicationsManagerAgent ApplicationsManagerAgent;
private PackageDatabaseAgent_DatabaseAgent DatabaseAgent;
private PackageDevelopmentAgent_DevelopmentAgent DevelopmentAgent;
private PackageGUIAgent_GUIAgent GUIAgent;
private PackageIntelligenceAgent_IntelligenceAgent IntelligenceAgent;
private PackageMessageAnalyzerAgent_MessageAnalyzerAgent MessageAnalyzerAgent;
private PackageMessengerAgent_MessengerAgent MessengerAgent;
private PackageSoftwareTransferAgent_SoftwareTransferAgent SoftwareTransferAgent;
private PackageSystemsAgent_SystemsAgent SystemsAgent;
private PackageVersionTrackingAgent_VersionTrackingAgent VersionTrackingAgent;
private PackageVirtualNetworksManagerAgent_VirtualNetworksManagerAgent VirtualNetworksManagerAgent;
private PackageColinAgent_ColinAgent ColinAgent;
//end of agent declarations
private List CreateNewAlgorithmVariableEventListeners;
private List DeleteAlgorithmVariableEventListeners;
private List ModifyAlgorithmVariableEventListeners;
private List ModifyAlgorithmListEventListeners;
private List InitiateAlgorithmEventListeners;
private List RegisterNewApplicationEventListeners;
private List UpdateExistingApplicationEventListeners;
private List DeleteExistingApplicationEventListeners;
private List RegisterNewApplicationServiceEventListeners;
private List UpdateExistingApplicationServiceEventListeners;
private List DeleteExistingApplicationServiceEventListeners;
private List SendAvailableNodesListToIntelligenceLayerEventListeners;
private List InquireAboutApplicationServicesEventListeners;
private List SearchForApplicationServiceEventListeners;
private List SendNodeWithFoundApplicationServiceToIntelligenceLayerEventListeners;
private List SendHolonicToIntelligenceEventListeners;
private List SendListOfVirtualNetworksEventListeners;
private List BackupQueueForSendingSplitUpMessagesTimerUpFunctionEventListeners;
private List DeleteBackupSplitUpMessageEventListeners;
private List SendIntelligenceToHolonicEventListeners;
private List SendHolonicToIntelligenceEventListeners;
private List SendQueueOfMenuOptionsToHTPMasterEventListeners;
private List ShowAgentMenusEventListeners;
private List BackupHolonicSystemNowEventListeners;
private List RestoreHolonicSystemNowEventListeners;
private List ReadClassLoadingEventListeners;
private List AdministrationConsoleEventListeners;
private List ApplicationsManagerConsoleEventListeners;
private List VirtualNetworksManagerConsoleEventListeners;
private List SystemsConsoleEventListeners;
private List SoftwareTransferConsoleEventListeners;
private List DevelopmentConsoleEventListeners;
private List VersionTrackingConsoleEventListeners;
private List DatabaseConsoleEventListeners;
private List AlgorithmsConsoleEventListeners;
systems event listeners
private List ReadCompilationSystemEventListeners;
private List ReadMemorySystemEventListeners;
private List ReadMemoryManagerEventListeners;
private List ReadMemoryPoolEventListeners;
private List ReadGarbageCollectorEventListeners;
private List ReadThreadingSystemEventListeners;
private List ReadRuntimeSystemEventListeners;
private List ReadOperatingSystemEventListeners;

private List CreateNewVirtualNetworkEventListeners;
private List DeleteExistingVirtualNetworkEventListeners;
private List RegisterNodeOnNetworkEventListeners;
private List CreateNewAvailableNodeEventListeners;
private List InformNewNodeOfOtherAvailableNodesEventListeners;
private List UpdateAvailableNodesListEventListeners;
private List RequestingAllAvailableNodesEventListeners;
private List CreateNewVirtualNetworkOnEntireNetworkEventListeners;
private List CreateVirtualLinkEventListeners;
private List SetUpVirtualNetworkEnvironmentEventListeners;
private List DeleteVirtualLinkEventListeners;
private List SetUpVirtualNetworkUsingTextFileEventListeners;
private List DeleteVirtualNetworkOnEntireNetworkEventListeners;

private List SendByteCodeEventListeners;
private List AddSegmentOfByteCodeEventListeners;
private List ConcludeSoftwareTransferEventListeners;
private List InitiateSoftwareUpdateEventListeners;
private List TestByteCodeEventListeners;
private List CopyAndRenameClassFileEventListeners;
private List SendApplicationSoftwareEventListeners;
private List SendAgentSoftwareEventListeners;
private List SendExternalMessageForIntelligenceAnalysisEventListeners;
private List SendIntelligenceResultsToMessengerAgentEventListeners;
private List CreateNewAgentCodeUsingTemplateEventListeners;
private List ListAgentWithSystemEventListeners;
private List MessageAnalysisEventListeners;
private List MainInputQueueEventListeners;
private List ApplicationsInputQueueEventListeners;
private List AdminApplicationsInputQueueEventListeners;
private List UserApplicationsInputQueueEventListeners;
private List VirtualNetworksInputQueueEventListeners;
private List AdminVirtualNetworksInputQueueEventListeners;
private List UserVirtualNetworksInputQueueEventListeners;

private List UserApplicationsOutputQueueEventListeners;
private List AdminApplicationsOutputQueueEventListeners;
private List ApplicationsOutputQueueEventListeners;
private List UserVirtualNetworksOutputQueueEventListeners;
private List AdminVirtualNetworksOutputQueueEventListeners;
private List VirtualNetworksOutputQueueEventListeners;
private List MainOutputQueueEventListeners;
private List BackupMainOutputQueueEventListeners;

II these listeners are for newly-designed internal message queues ***
//lists-of-internal-listeners declarations
private List DemonstrationAgentInternalMessageListeners;
private List ColinAgentInternalMessageListeners;
private List HandleInternalMessagesListeners;
private List AdministrationAgentInternalMessageListeners;
private List AlgorithmsAgentInternalMessageListeners;
private List ApplicationsManagerAgentInternalMessageListeners;
private List ConsoleAgentInternalMessageListeners;
private List DatabaseAgentInternalMessageListeners;
private List DevelopmentAgentInternalMessageListeners;
private List GUIAgentInternalMessageListeners;
private List IntelligenceAgentInternalMessageListeners;
private List MessageAnalyzerAgentInternalMessageListeners;
private List MessengerAgentInternalMessageListeners;
private List SoftwareTransferAgentInternalMessageListeners;
private List SystemsAgentInternalMessageListeners;
private List VersionTrackingAgentInternalMessageListeners;
private List VirtualNetworksManagerAgentInternalMessageListeners;

private HolonicTechnologyPlatformMaster HolonicTechnologyPlatformMaster;

/**************** /
/* constructors */
/**************** /

public HolonicTechnologyPlatformMaster(ExecuteHolonicTechnologyPlatform
InputLaunchingPlatform)
{
LaunchingPlatform = InputLaunchingPlatform;

//setting boolean conditions
Stop = false;
StopDisplayingConsoleMenu = false;
FirstTimeDisplayingConsoleMenu = false;
SystemLocalTime = 0;
SystemTimeStamp = 0;
UpdatedCalendar = new GregorianCalendar();
NumberOfCoreAgents = 0;
NumberOfCoreAgentsAsString = "";
NumberOfPropAgents = 0;
NumberOfPropAgentsAsString = "";
//AtomicSynchronizationTimer = new Timer();

//attempting to set the OwnIPAddress and OwnIPNumber variables
try {
    OwnIPAddress = InetAddress.getLocalHost();
    String ownIPAddressToString = OwnIPAddress.toString();
    int positionMarker = ownIPAddressToString.indexOf("/");
    ComputerName = ownIPAddressToString.substring(0, positionMarker);
    OwnIPNumber = ownIPAddressToString.substring(positionMarker + 1);
}
catch(UnknownHostException e) {
    System.out.println("Unable to get own IP-address");
    System.out.println(e);
}

//setting port numbers
IncomingCommunicationsPort = 4686;
OutgoingCommunicationsPort = 4683;
IncomingApplicationsPort = 4692;
OutgoingApplicationsPort = 4689;
IncomingHolonicPortOnApplicationsLayer = 4695;

/*
//setting communications socket connection
try {
    IncomingCommunicationsSocket = new DatagramSocket(IncomingCommunicationsPort);
}
catch(IOException e) {
    System.out.println(e);
}

//setting applications socket connection
try {
    IncomingApplicationsSocket = new DatagramSocket(IncomingApplicationsPort);
}
catch(IOException e) {
    System.out.println(e);
}
ListOfAgents = new LinkedList();
MainInputQueue = new PackageAbstractComponents_AbstractInputOutputQueue();
MainOutputQueue = new PackageAbstractComponents_AbstractInputOutputQueue();
BackupMainOutputQueue = new PackageAbstractComponents_AbstractInputOutputQueue();
CurrentMessageCountValue = 0;

//setting up event listeners
CreateNewAlgorithmVariableEventListeners = new ArrayList();
DeleteAlgorithmVariableEventListeners = new ArrayList();
ModifyAlgorithmVariableEventListeners = new ArrayList();
ModifyAlgorithmListEventListeners = new ArrayList();
InitiateAlgorithmEventListeners = new ArrayList();
RegisterNewApplicationEventListeners = new ArrayList();
UpdateExistingApplicationEventListeners = new ArrayList();
DeleteExistingApplicationEventListeners = new ArrayList();
RegisterNewApplicationServiceEventListeners = new ArrayList();
UpdateExistingApplicationServiceEventListeners = new ArrayList();
DeleteExistingApplicationServiceEventListeners = new ArrayList();
SendAvailableNodesListToIntelligenceLayerEventListeners = new ArrayList();
InquireAboutApplicationServicesEventListeners = new ArrayList();
SearchForApplicationServiceEventListeners = new ArrayList();
SendNodeWithFoundApplicationServiceToIntelligenceLayerEventListeners = new ArrayList();
SendIntelligenceToIntelligenceEventListeners = new ArrayList();
SendHolonicstoIntelligenceEventListeners = new ArrayList();
SendListONetworksEventListeners = new ArrayList();
BackupQueueForSendingSplitUpMessagesTimerUpFunctionEventListeners = new ArrayList();
DeleteBackupSplitUpMessageEventListeners = new ArrayList();
SendIntelligenceToIntelligenceViaHolonicEventListeners = new ArrayList();
SendHolonicstoIntelligenceViaHolonicEventListeners = new ArrayList();
CreateNewVirtualNetworkEventListeners = new ArrayList();
DeleteExistingVirtualNetworkEventListeners = new ArrayList();
RegisterNodeOnNetworkEventListeners = new ArrayList();
CreateNewAvailableNodeEventListeners = new ArrayList();
InformNewNodeOfOtherAvailableNodesEventListeners = new ArrayList();
UpdateAvailableNodesListEventListeners = new ArrayList();
RequestingAllAvailableNodesEventListeners = new ArrayList();
CreateNewVirtualNetworkOnEntireNetworkEventListeners = new ArrayList();
CreateVirtualLinkEventListeners = new ArrayList();
SetUpVirtualNetworkEnvironmentEventListeners = new ArrayList();
DeleteVirtualLinkEventListeners = new ArrayList();
SetUpVirtualNetworkUsingTextFileEventListeners = new ArrayList();
DeleteVirtualNetworkOnEntireNetworkEventListeners = new ArrayList();
InitiateSaveAndQuitEventListeners = new ArrayList();
354  OutputToConsoleEventListeners = new ArrayList();
355  AdministrationConsoleEventListeners = new ArrayList();
356  SoftwareTransferConsoleEventListeners = new ArrayList();
357  DevelopmentConsoleEventListeners = new ArrayList();
358  VersionTrackingConsoleEventListeners = new ArrayList();
359  DatabaseConsoleEventListeners = new ArrayList();
360  AlgorithmsConsoleEventListeners = new ArrayList();
361  ShowAgentMenusEventListeners = new ArrayList();
362
363  //done only because this event is called at startup
364  //
365  SendQueueOfMenuOptionsToHTPMasterEventListeners = new ArrayList();
366  PackageConsoleAgent_SendQueueOfMenuOptionsToHTPMasterEventListener
367  SendQueueOfMenuOptionsToHTPMasterEventListener = new
368  PackageConsoleAgent_SendQueueOfMenuOptionsToHTPMasterEventHandler( );
369  this.AddSendQueueOfMenuOptionsToHTPMasterEventListener(SendQueueOfMenuOp
370  tionsToHTPMasterEventListener);
371
372
373  BackupHolonicSystemNowEventListeners = new ArrayList();
374  RestoreHolonicSystemNowEventListeners = new ArrayList();
375
376  ReadClassLoadingEventListeners = new ArrayList();
377  ReadCompilationSystemEventListeners = new ArrayList();
378  ReadMemorySystemEventListeners = new ArrayList();
379  ReadMemoryManagerEventListeners = new ArrayList();
380  ReadMemoryPoolEventListeners = new ArrayList();
381  ReadGarbageCollectorEventListeners = new ArrayList();
382  ReadThreadingSystemEventListeners = new ArrayList();
383  ReadRuntimeSystemEventListeners = new ArrayList();
384  ReadOperatingSystemEventListeners = new ArrayList();
385
386  SendByteCodeEventListeners = new ArrayList();
387  AddSegmentOfByteCodeEventListeners = new ArrayList();
388  ConcludeSoftwareTransferEventListeners = new ArrayList();
389  InitiateSoftwareUpdateEventListeners = new ArrayList();
390  TestByteCodeEventListeners = new ArrayList();
391  CopyAndRenameClassFileEventListeners = new ArrayList();
392  SendApplicationSoftwareEventListeners = new ArrayList();
393  SendAgentSoftwareEventListeners = new ArrayList();
394
395  SendExternalMessageForIntelligenceAnalysisEventListeners = new ArrayList();
396  SendIntelligenceResultsToMessengerAgentEventListeners = new ArrayList();
397
398  CreateNewAgentCodeUsingTemplateEventListeners = new ArrayList();
399  CreateNewApplicationAgentCodeUsingTemplateEventListeners = new ArrayList();
400  ListAgentWithSystemEventListeners = new ArrayList();
401  ListAgentWithSystemEventListeners = new ArrayList();
MessageAnalysisEventListeners = new ArrayList();
MainInputQueueEventListeners = new ArrayList();
MainOutputQueueEventListeners = new ArrayList();
ApplicationsInputQueueEventListeners = new ArrayList();
ApplicationsOutputQueueEventListeners = new ArrayList();
AdminApplicationsInputQueueEventListeners = new ArrayList();
AdminApplicationsOutputQueueEventListeners = new ArrayList();
UserApplicationsInputQueueEventListeners = new ArrayList();
UserApplicationsOutputQueueEventListeners = new ArrayList();
BackupMainOutputQueueEventListeners = new ArrayList();
UserVirtualNetworksOutputQueueEventListeners = new ArrayList();
AdminVirtualNetworksOutputQueueEventListeners = new ArrayList();
VirtualNetworksOutputQueueEventListeners = new ArrayList();
UserVirtualNetworksInputQueueEventListeners = new ArrayList();
AdminVirtualNetworksInputQueueEventListeners = new ArrayList();
VirtualNetworksInputQueueEventListeners = new ArrayList();

// initialization of internal message listener lists
DemonstrationAgentInternalMessageListeners = new ArrayList();
ColinAgentInternalMessageListeners = new ArrayList();
HandleInternalMessagesListeners = new ArrayList();
AdministrationAgentInternalMessageListeners = new ArrayList();
AlgorithmsAgentInternalMessageListeners = new ArrayList();
ApplicationsManagerAgentInternalMessageListeners = new ArrayList();
ConsoleAgentInternalMessageListeners = new ArrayList();
DatabaseAgentInternalMessageListeners = new ArrayList();
DevelopmentAgentInternalMessageListeners = new ArrayList();
IntelligenceAgentInternalMessageListeners = new ArrayList();
MessageAnalyzerAgentInternalMessageListeners = new ArrayList();
MessengerAgentInternalMessageListeners = new ArrayList();
SoftwareTransferAgentInternalMessageListeners = new ArrayList();
SystemsAgentInternalMessageListeners = new ArrayList();
VersionTrackingAgentInternalMessageListeners = new ArrayList();
VirtualNetworksManagerAgentInternalMessageListeners = new ArrayList();

// setting up agents
ConsoleAgent = new PackageConsoleAgent_ConsoleAgent(this);
AdministrationAgent = new PackageAdministrationAgent_AdministrationAgent(this);
AlgorithmsAgent = new PackageAlgorithmsAgent_AlgorithmsAgent(this);
ApplicationsManagerAgent = new PackageApplicationsManagerAgent_ApplicationsManagerAgent(this);
DatabaseAgent = new PackageDatabaseAgent_DatabaseAgent(this);
DevelopmentAgent = new PackageDevelopmentAgent_DevelopmentAgent(this);
GUIAgent = new PackageGUIAgent_GUIAgent(this);
IntelligenceAgent = new PackageIntelligenceAgent_IntelligenceAgent(this);
MessageAnalyzerAgent = new PackageMessageAnalyzerAgent_MessageAnalyzerAgent(this);
MessengerAgent = new PackageMessengerAgent_MessengerAgent(this);
SoftwareTransferAgent = new PackageSoftwareTransferAgent_SoftwareTransferAgent(this);
462  SystemsAgent = new PackageSystemsAgent_SystemsAgent(this);
463  VersionTrackingAgent = new
464  PackageVersionTrackingAgent_VersionTrackingAgent(this);
465  VirtualNetworksManagerAgent = new
466  PackageVirtualNetworksManagerAgent_VirtualNetworksManagerAgent(this);
467  ColinAgent = new PackageColinAgent_ColinAgent(this);
468  // end of setting up agents
469
470  /*
471     * synchronise time with atomic clock every 10 seconds
472     */
473  this.BeginAtomicSynchronizationTimer();
474
475  /*
476     * start autosave timer
477     */
478  this.BeginAutosaveTimer();
479
480  /
481  /*
482     * adding list of agents to team name "CORE" - CORE will
483     * be the core development agents
484     */
485  /
486  // create new team
487  PackageDevelopmentAgent_AgentTeam CoreAgentTeam = new
488  PackageDevelopmentAgent_AgentTeam("CORE");
489
490  // create team members and add to new team
491  PackageDevelopmentAgent_AgentTeamMember CoreAgentTeamMember1
492  = new PackageDevelopmentAgent_AgentTeamMember("Administration Agent",
493  "CORE");
494  CoreAgentTeam.AddAgentTeamMember(CoreAgentTeamMember1);
495  NumberOfCoreAgents++;
496
497  PackageDevelopmentAgent_AgentTeamMember CoreAgentTeamMember2
498  = new PackageDevelopmentAgent_AgentTeamMember("Applications Manager
499  Agent","CORE");
500  CoreAgentTeam.AddAgentTeamMember(CoreAgentTeamMember2);
501  NumberOfCoreAgents++;
502
503  PackageDevelopmentAgent_AgentTeamMember CoreAgentTeamMember3
504  = new PackageDevelopmentAgent_AgentTeamMember("Console Agent",
505  "CORE");
506  CoreAgentTeam.AddAgentTeamMember(CoreAgentTeamMember3);
507  NumberOfCoreAgents++;
508
509  PackageDevelopmentAgent_AgentTeamMember CoreAgentTeamMember4
510  = new PackageDevelopmentAgent_AgentTeamMember("Database Agent",
511  "CORE");
512  CoreAgentTeam.AddAgentTeamMember(CoreAgentTeamMember4);
513  NumberOfCoreAgents++;
PackageDevelopmentAgent_AgentTeamMember CoreAgentTeamMember5
= new PackageDevelopmentAgent_AgentTeamMember("Development Agent", "CORE");
CoreAgentTeam.AddAgentTeamMember(CoreAgentTeamMember5);
NumberOfCoreAgents++;

PackageDevelopmentAgent_AgentTeamMember CoreAgentTeamMember6
= new PackageDevelopmentAgent_AgentTeamMember("Message Analyzer Agent", "CORE");
CoreAgentTeam.AddAgentTeamMember(CoreAgentTeamMember6);
NumberOfCoreAgents++;

PackageDevelopmentAgent_AgentTeamMember CoreAgentTeamMember7
= new PackageDevelopmentAgent_AgentTeamMember("Messenger Agent", "CORE");
CoreAgentTeam.AddAgentTeamMember(CoreAgentTeamMember7);
NumberOfCoreAgents++;

PackageDevelopmentAgent_AgentTeamMember CoreAgentTeamMember8
= new PackageDevelopmentAgent_AgentTeamMember("Software Transfer Agent", "CORE");
CoreAgentTeam.AddAgentTeamMember(CoreAgentTeamMember8);
NumberOfCoreAgents++;

PackageDevelopmentAgent_AgentTeamMember CoreAgentTeamMember9
= new PackageDevelopmentAgent_AgentTeamMember("Systems Agent", "CORE");
CoreAgentTeam.AddAgentTeamMember(CoreAgentTeamMember9);
NumberOfCoreAgents++;

PackageDevelopmentAgent_AgentTeamMember CoreAgentTeamMember10
= new PackageDevelopmentAgent_AgentTeamMember("Version Tracking Agent", "CORE");
CoreAgentTeam.AddAgentTeamMember(CoreAgentTeamMember10);
NumberOfCoreAgents++;

PackageDevelopmentAgent_AgentTeamMember CoreAgentTeamMember11
= new PackageDevelopmentAgent_AgentTeamMember("Virtual Networks ManagerAgent", "CORE");
CoreAgentTeam.AddAgentTeamMember(CoreAgentTeamMember11);
NumberOfCoreAgents++;

PackageDevelopmentAgent_AgentTeamMember CoreAgentTeamMember12
= new PackageDevelopmentAgent_AgentTeamMember("Algorithms Agent", "CORE");
CoreAgentTeam.AddAgentTeamMember(CoreAgentTeamMember12);
NumberOfCoreAgents++;

PackageDevelopmentAgent_AgentTeamMember CoreAgentTeamMember13
= new PackageDevelopmentAgent_AgentTeamMember("GUI Agent", "CORE");
CoreAgentTeam.AddAgentTeamMember(CoreAgentTeamMember13);
NumberOfCoreAgents++;
CoreAgentTeamMember14
= new PackageDevelopmentAgent_AgentTeamMember("Intelligence Agent", "CORE");
CoreAgentTeam.AddAgentTeamMember(CoreAgentTeamMember14);
NumberOfCoreAgents++;
//total number of core agents:
//14
NumberofCoreAgentsAsString = Integer.toString(NumberOfCoreAgents);
//add team
this.GetDevelopmentAgent().GetDatabaseMaster().AddAgentTeam(CoreAgentTeam);

/*
 * adding list of agents to team name "PROP" - PROP will be the proprietary development agents
 */

//create new team
PackageDevelopmentAgent_AgentTeam PropAgentTeam = new PackageDevelopmentAgent_AgentTeam("PROP");

PackageDevelopmentAgent_AgentTeamMember
   PropAgentTeamMember1
= new PackageDevelopmentAgent_AgentTeamMember("Colin Agent", "PROP");
PropAgentTeam.AddAgentTeamMember(PropAgentTeamMember1);
NumberOfPropAgents++;
//total number of prop agents:
//1
NumberOfPropAgentsAsString = Integer.toString(NumberOfPropAgents);
//add team
this.GetDevelopmentAgent().GetDatabaseMaster().AddAgentTeam(PropAgentTeam);

//setting up available host node
PackageVirtualNetworksManagerAgent_QueueOfAvailableNodes
   TempQueueOfAvailableNodes =
   VirtualNetworksManagerAgent.GetDatabaseMaster().GetQueueOfAvailableNodes();

PackageVirtualNetworksManagerAgent_AvailableNode
   HostNode =
   new PackageVirtualNetworksManagerAgent_AvailableNode();
HostNode.SetNodeIPNumber(OwnIPNumber);
HostNode.SetNodeIPAddress(OwnIPAddress);
HostNode.SetNodeName(ComputerName);

PackageVirtualNetworksManagerAgent_AvailableNode
   DummyAvailableNode = TempQueueOfAvailableNodes.ModifyQueue(1, HostNode);
public void Destructor()
{
    LaunchingPlatform = null;

    //setting boolean conditions
    Stop = false;
    StopDisplayingConsoleMenu = false;
    FirstTimeDisplayingConsoleMenu = false;
    SystemLocalTime = 0;
    SystemTimeStamp = 0;
    UpdatedCalendar = null;
    NumberOfCoreAgents = 0;
    NumberOfCoreAgentsAsString = null;
    NumberOfPropAgents = 0;
    NumberOfPropAgentsAsString = null;
    OwnlPAddress = null;
    ComputerName = null;
    OwnlPNumber = null;

    //setting port numbers
    IncomingCommunicationsPort = 0;
    OutgoingCommunicationsPort = 0;
    IncomingApplicationsPort = 0;
    OutgoingApplicationsPort = 0;
    IncomingHolonicPortOnApplicationsLayer = 0;

    //setting communications socket connection
    In/IncomingCommunicationsSocket = null;

    //setting applications socket connection
    In/IncomingApplicationsSocket = null;

    ListOfAgents = null;

    MainInputQueue = null;
    MainOutputQueue = null;
    BackupMainOutputQueue = null;
    CurrentMessageCountValue = 0;

    //setting up agents
    AdministrationAgent = null;
    AlgorithmsAgent = null;
    ApplicationsManagerAgent = null;
    ConsoleAgent = null;
    DatabaseAgent = null;
    DevelopmentAgent = null;
    GUIAgent = null;
    IntelligenceAgent = null;
    MessageAnalyzerAgent = null;
MessengerAgent = null;
SoftwareTransferAgent = null;
SystemsAgent = null;
VersionTrackingAgent = null;
VirtualNetworksManagerAgent = null;
ColinAgent = null;
//end of agent destruction

IIAtomicSynchronizationTimer.canceIO;
IIAtomicSynchronizationTimer = null;

CreateNewAlgorithmVariableEventListeners = null;
DeleteAlgorithmVariableEventListeners = null;
ModifyAlgorithmVariableEventListeners = null;
ModifyAlgorithmListEventListeners = null;
InitiateAlgorithmEventListeners = null;

RegisterNewApplicationEventListeners = null;
UpdateExistingApplicationEventListeners = null;
DeleteExistingApplicationEventListeners = null;
RegisterNewApplicationServiceEventListeners = null;
UpdateExistingApplicationServiceEventListeners = null;
DeleteExistingApplicationServiceEventListeners = null;
SendAvailableNodesListToIntelligenceLayerEventListeners = null;
InquireAboutApplicationServicesEventListeners = null;
SearchForApplicationServiceEventListeners = null;
SendNodeWithFoundApplicationServiceToIntelligenceLayerEventListeners = null;
SendIntelligenceToIntelligenceEventListeners = null;
SendIntelligenceForIntelligenceEventListeners = null;
SendListONetworksEventListeners = null;
BackupQueueForSendingSplitUpMessagesTimerUpFunctionEventListeners = null;
DeleteBackupSplitUpMessageEventListeners = null;
SendIntelligenceToIntelligenceViaHolonicEventListeners = null;
SendHolonicToIntelligenceEventListeners = null;

CreateNewVirtualNetworkEventListeners = null;
DeleteExistingVirtualNetworkEventListeners = null;
RegisterNodeOnNetworkEventListeners = null;
CreateNewAvailableNodeEventListeners = null;
InformNewNodeOfOtherAvailableNodesEventListeners = null;
UpdateAvailableNodesListEventListeners = null;
RequestingAllAvailableNodesEventListeners = null;
CreateNewVirtualNetworkOnEntireNetworkEventListeners = null;
CreateVirtualLinkEventListeners = null;
SetUpVirtualNetworkEnvironmentEventListeners = null;
DeleteVirtualLinkEventListeners = null;
SetUpVirtualNetworkUsingTextFileEventListeners = null;
DeleteVirtualNetworkOnEntireNetworkEventListeners = null;

InitiateSaveAndQuitEventListeners = null;
OutputToConsoleEventListeners = null;
AdministrationConsoleEventListeners = null;
VirtualNetworksManagerConsoleEventListeners = null;
ApplicationsManagerConsoleEventListeners = null;
SystemsConsoleEventListeners = null;
SoftwareTransferConsoleEventListeners = null;
DevelopmentConsoleEventListeners = null;
VersionTrackingConsoleEventListeners = null;
DatabaseConsoleEventListeners = null;
AlgorithmsConsoleEventListeners = null;
BackupHolonicSystemNowEventListeners = null;
RestoreHolonicSystemNowEventListeners = null;
ReadClassLoadingEventListeners = null;
ReadCompilationSystemEventListeners = null;
ReadMemorySystemEventListeners = null;
ReadMemoryManagerEventListeners = null;
ReadMemoryPoolEventListeners = null;
ReadGarbageCollectorEventListeners = null;
//ReadThreadingSystemEventListeners = null;
ReadRuntimeSystemEventListeners = null;
ReadOperatingSystemEventListeners = null;
SendByteCodeEventListeners = null;
AddSegmentOfByteCodeEventListeners = null;
ConcludeSoftwareTransferEventListeners = null;
InitiateSoftwareUpdateEventListeners = null;
TestByteCodeEventListeners = null;
CopyAndRenameClassFileEventListeners = null;
SendApplicationSoftwareEventListeners = null;
SendAgentSoftwareEventListeners = null;
SendExternalMessageForIntelligenceAnalysisEventListeners = null;
CreateNewAgentCodeUsingTemplateEventListeners = null;
CreateNewApplicationAgentCodeUsingTemplateEventListeners = null;
ListAgentWithSystemEventListeners = null;
MessageAnalysisEventListeners = null;
MainInputQueueEventListeners = null;
MainOutputQueueEventListeners = null;
ApplicationsInputQueueEventListeners = null;
ApplicationsOutputQueueEventListeners = null;
AdminApplicationsInputQueueEventListeners = null;
AdminApplicationsOutputQueueEventListeners = null;
UserApplicationsInputQueueEventListeners = null;
UserApplicationsOutputQueueEventListeners = null;
BackupMainOutputQueueEventListeners = null;
UserVirtualNetworksOutputQueueEventListeners = null;
AdminVirtualNetworksOutputQueueEventListeners = null;
VirtualNetworksOutputQueueEventListeners = null;
UserVirtualNetworksInputQueueEventListeners = null;
AdminVirtualNetworksInputQueueEventListeners = null;
VirtualNetworksInputQueueEventListeners = null;
//deconstruction of internal message listener lists
ColinAgentInternalMessageListeners = null;
HandleInternalMessagesListeners = null;
AdministrationAgentInternalMessageListeners = null;
AlgorithmsAgentInternalMessageListeners = null;
ApplicationsManagerAgentInternalMessageListeners = null;
ConsoleAgentInternalMessageListeners = null;
DatabaseAgentInternalMessageListeners = null;
DevelopmentAgentInternalMessageListeners = null;
IntelligenceAgentInternalMessageListeners = null;
GUIAgentInternalMessageListeners = null;
MessageAnalyzerAgentInternalMessageListeners = null;
MessengerAgentInternalMessageListeners = null;
SoftwareTransferAgentInternalMessageListeners = null;
SystemsAgentInternalMessageListeners = null;
VersionTrackingAgentInternalMessageListeners = null;
VirtualNetworksManagerAgentInternalMessageListeners = null;
HolonicTechnologyPlatformMaster = null;

/************* /
/* accessors */
/************* /

public ExecuteHolonicTechnologyPlatform GetLaunchingPlatform()
{
    return LaunchingPlatform;
}

public InetAddress GetOwnIPAddress()
{
    return OwnIPAddress;
}

public String GetOwnIPNumber()
{
    return OwnIPNumber;
}

public long GetSystemLocalTime()
{
    return SystemLocalTime;
}

public long GetSystemTimeStamp()
{
    return SystemTimeStamp;
}

public int GetIncomingHolonicPortOnApplicationsLayer()
830   return IncomingHolonicPortOnApplicationsLayer;
831 }
832
833
834 public int GetIncomingCommunicationsPort()
835 {  
836   return IncomingCommunicationsPort;
837 }
838
839
840 public int GetOutgoingCommunicationsPort()
841 {  
842   return OutgoingCommunicationsPort;
843 }
844
845
846 public int GetIncomingApplicationsPort()
847 {  
848   return IncomingApplicationsPort;
849 }
850
851
852 public int GetOutgoingApplicationsPort()
853 {  
854   return OutgoingApplicationsPort;
855 }
856
857
858 public String GetComputerName()
859 { 
860   return ComputerName;
861 }
862
863
864 public LinkedList GetListOfAgents()
865 { 
866   return ListOfAgents;
867 }
868
869
870 public PackageAbstractComponents_AbstractInputOutputQueue GetMainInputQueue()
871 { 
872   return MainInputQueue;
873 }
874
875
876 public PackageAbstractComponents_AbstractInputOutputQueue GetMainOutputQueue()
877 { 
878   return MainOutputQueue;
879 }
880
881
882 public PackageAbstractComponents_AbstractInputOutputQueue GetBackupMainOutputQueue()
883 {
884   return BackupMainOutputQueue;
885 }
public PackageAdministrationAgent_AdministrationAgent GetAdministrationAgent() {
  return AdministrationAgent;
}

public PackageAlgorithmsAgent_AlgorithmsAgent GetAlgorithmsAgent() {
  return AlgorithmsAgent;
}

public PackageApplicationsManagerAgent_ApplicationsManagerAgent GetApplicationsManagerAgent() {
  return ApplicationsManagerAgent;
}

public PackageConsoleAgent_ConsoleAgent GetConsoleAgent() {
  return ConsoleAgent;
}

public PackageDatabaseAgent_DatabaseAgent GetDatabaseAgent() {
  return DatabaseAgent;
}

public PackageDevelopmentAgent_DevelopmentAgent GetDevelopmentAgent() {
  return DevelopmentAgent;
}

public PackageGUIAgent_GUIAgent GetGUIAgent() {
  return GUIAgent;
}

public PackageIntelligenceAgent_IntelligenceAgent GetIntelligenceAgent() {
  return IntelligenceAgent;
}

public PackageMessageAnalyzerAgent_MessageAnalyzerAgent GetMessageAnalyzerAgent() {
  return MessageAnalyzerAgent;
}
public PackageMessengerAgent_MessengerAgent GetMessengerAgent() {
    return MessengerAgent;
}

public PackageSoftwareTransferAgent_SoftwareTransferAgent GetSoftwareTransferAgent() {
    return SoftwareTransferAgent;
}

public PackageSystemsAgent_SystemsAgent GetSystemsAgent() {
    return SystemsAgent;
}

public PackageVirtualNetworksManagerAgent_VirtualNetworksManagerAgent GetVirtualNetworksManagerAgent() {
    return VirtualNetworksManagerAgent;
}

public PackageVersionTrackingAgent_VersionTrackingAgent GetVersionTrackingAgent() {
    return VersionTrackingAgent;
}

public PackageColinAgent_ColinAgent GetColinAgent() {
    return ColinAgent;
}

public synchronized void AddHandleInternalMessagesEventListener(PackageExecutable_SwitchMechanismHandleInternalMessagesEventListener Listener) {
    HandleInternalMessagesListeners.add(Listener);
}

public synchronized void RemoveHandleInternalMessagesEventListener(PackageExecutable_SwitchMechanismHandleInternalMessagesEventListener Listener) {
    HandleInternalMessagesListeners.remove(Listener);
}

public synchronized void SetHandleInternalMessagesEventListener(PackageExecutable_SwitchMechanismHandleInternalMessagesEventListener Listener) {
    HandleInternalMessagesListeners.set(Listener);
}
public synchronized void FireHandleInternalMessagesEvent(
PackageAbstractComponents_AbstractInternalMessage InternalMessageBeingSent)
{
//creates new event and fires it to be handled
PackageExecutable_SwitchMechanismHandleInternalMessagesEvent NewEvent =
new PackageExecutable_SwitchMechanismHandleInternalMessagesEvent(
this, this, InternalMessageBeingSent);
Iterator Listeners = HandleInternalMessagesListeners.iterator();
boolean test = Listeners.hasNext();
while(Listeners.hasNext())
{
((PackageExecutable_SwitchMechanismHandleInternalMessagesEventListener)
Listeners.next()).HandleInternalMessagesEventReceived(NewEvent);
}
this.IncrementSystemLocalTime();
//creates new event and fires it to be handled
PackageExecutable_SwitchMechanismHandleInternalMessagesEvent NewEvent =
new PackageExecutable_SwitchMechanismHandleInternalMessagesEvent(
this, this, InternalMessageBeingSent);
Iterator Listeners = HandleInternalMessagesListeners.iterator();
boolean test = Listeners.hasNext();
while(Listeners.hasNext())
{
((PackageExecutable_SwitchMechanismHandleInternalMessagesEventListener)
Listeners.next()).HandleInternalMessagesEventReceived(NewEvent);
}

//FireServiceAdministrationAgentInternalMessageQueueEvent
public synchronized void AddServiceAdministrationAgentInternalMessageQueueEventListener(
PackageAdministrationAgent_ServiceInternalMessagesEventListener Listener)
{
AdministrationAgentInternalMessageListeners.add(Listener);
}

public synchronized void RemoveServiceAdministrationAgentInternalMessageQueueEventListener(
PackageAdministrationAgent_ServiceInternalMessagesEventListener Listener)
{
AdministrationAgentInternalMessageListeners.remove(Listener);
}

public synchronized void FireServiceAdministrationAgentInternalMessageQueueEvent()
{
//creates new event and fires it to be handled
PackageAdministrationAgent_ServiceInternalMessagesEvent NewEvent =
new PackageAdministrationAgent_ServiceInternalMessagesEvent(
this, this);
Iterator Listeners = AdministrationAgentInternalMessageListeners.iterator();
boolean test = Listeners.hasNext();
while(Listeners.hasNext())
{
((PackageAdministrationAgent_ServiceInternalMessagesEventListener)
Listeners.next()).ServiceInternalMessagesEventReceived(NewEvent);
}
\begin{verbatim}
1043  this.incrementSystemLocalTime();
1044 }
1045
1047 //FireServiceAlgorithmsAgentInternalMessageQueueEvent
1048 public synchronized void
1049     AddServiceAlgorithmsAgentInternalMessageQueueEventListener(
1050     AlgorithmsAgent_ServiceInternalMessagesEventListener Listener)
1051 {
1052     AlgorithmsAgentInternalMessageListeners.add(Listener);
1053 }
1054
1055 public synchronized void
1056     RemoveServiceAlgorithmsAgentInternalMessageQueueEventListener(
1057     AlgorithmsAgent_ServiceInternalMessagesEventListener Listener)
1058 {
1059     AlgorithmsAgentInternalMessageListeners.remove(Listener);
1060 }
1061
1062 //FireServiceAlgorithmsAgentInternalMessageQueueEvent
1063 public synchronized void
1064     FireServiceAlgorithmsAgentInternalMessageQueueEvent()
1065 {
1066     //creates new event and fires it to be handled
1067     PackageAlgorithmsAgent_ServiceInternalMessagesEvent NewEvent =
1068         new PackageAlgorithmsAgent_ServiceInternalMessagesEvent(
1069             this, this);
1070     Iterator Listeners = AlgorithmsAgentInternalMessageListeners.iterator();
1071     boolean test = Listeners.hasNext();
1072     while(Listeners.hasNext())
1073         {((PackageAlgorithmsAgent_ServiceInternalMessagesEventListener)Listeners.
1074             next()).ServiceInternalMessagesEventReceived(NewEvent);
1075         }
1076     this.incrementSystemLocalTime();
1077 }
1078
1079 //FireServiceApplicationsManagerAgentInternalMessageQueueEvent
1080 public synchronized void
1081     AddServiceApplicationsManagerAgentInternalMessageQueueEventListener(
1082     PackageApplicationsManagerAgent_ServiceInternalMessagesEventListener Listener)
1083 {
1084     ApplicationsManagerAgentInternalMessageListeners.add(Listener);
1085 }
1086
1087 public synchronized void
1088     RemoveServiceApplicationsManagerAgentInternalMessageQueueEventListener(
1089     PackageApplicationsManagerAgent_ServiceInternalMessagesEventListener Listener)
1090 {
\end{verbatim}
ApplicationsManagerAgentInternalMessageListeners.remove(Listener);

public synchronized void FireServiceApplicationsManagerAgentInternalMessageQueueEvent() {
    //creates new event and fires it to be handled
    PackageApplicationsManagerAgent_ServiceInternalMessagesEvent NewEvent =
        new PackageApplicationsManagerAgent_ServiceInternalMessagesEvent(
            this, this);
    Iterator Listeners = ApplicationsManagerAgentInternalMessageListeners.iterator();
    boolean test = Listeners.hasNext();
    while(Listeners.hasNext()) {
        ((PackageApplicationsManagerAgent_ServiceInternalMessagesEventListener)L
            Listeners.next()).ServiceInternalMessagesEventReceived(NewEvent);
    }
    this.incrementSystemLocalTime();
}

public synchronized void FireServiceConsoleAgentInternalMessageQueueEvent() {
    //creates new event and fires it to be handled
    PackageConsoleAgent_ServiceInternalMessagesEvent NewEvent =
        new PackageConsoleAgent_ServiceInternalMessagesEvent(
            this, this);
    Iterator Listeners = ConsoleAgentInternalMessageListeners.iterator();
    boolean test = Listeners.hasNext();
    while(Listeners.hasNext()) {
        ((PackageConsoleAgent_ServiceInternalMessagesEventListener)L
            Listeners.next()).ServiceInternalMessagesEventReceived(NewEvent);
    }
    this.incrementSystemLocalTime();
}

public synchronized void AddServiceConsoleAgentInternalMessageQueueEventListener(
    PackageConsoleAgent_ServiceInternalMessagesEventListener Listener) {
    ConsoleAgentInternalMessageListeners.add(Listener);
}

public synchronized void RemoveServiceConsoleAgentInternalMessageQueueEventListener(
    PackageConsoleAgent_ServiceInternalMessagesEventListener Listener) {
    ConsoleAgentInternalMessageListeners.remove(Listener);
}

public synchronized void FireServiceConsoleAgentInternalMessageQueueEvent() {
    //creates new event and fires it to be handled
    PackageConsoleAgent_ServiceInternalMessagesEvent NewEvent =
        new PackageConsoleAgent_ServiceInternalMessagesEvent(
            this, this);
    Iterator Listeners = ConsoleAgentInternalMessageListeners.iterator();
    boolean test = Listeners.hasNext();
    while(Listeners.hasNext()) {
        ((PackageConsoleAgent_ServiceInternalMessagesEventListener)L
            Listeners.next()).ServiceInternalMessagesEventReceived(NewEvent);
    }
}
DatabaseAgentInternalMessageListeners.add(Listener);

DatabaseAgentInternalMessageListeners.remove(Listener);

public synchronized void RemoveServiceDatabaseAgentInternalMessageQueueEventListener(PackageDatabaseAgent_ServiceInternalMessagesEventListener Listener)
{
    this.incrementSystemLocalTime();
}

//FireServiceDatabaseAgentInternalMessageQueueEvent
public synchronized void
    AddServiceDatabaseAgentInternalMessageQueueEventListener(PackageDatabaseAgent_ServiceInternalMessagesEventListener Listener)
{
    DatabaseAgentInternalMessageListeners.add(Listener);
}

public synchronized void
    RemoveServiceDatabaseAgentInternalMessageQueueEventListener(PackageDatabaseAgent_ServiceInternalMessagesEventListener Listener)
{
    DatabaseAgentInternalMessageListeners.remove(Listener);
}

public synchronized void
    FireServiceDatabaseAgentInternalMessageQueueEvent()
{
    //creates new event and fires it to be handled
    PackageDatabaseAgent_ServiceInternalMessagesEvent NewEvent =
        new PackageDatabaseAgent_ServiceInternalMessagesEvent(
            this, this);
    Iterator Listeners = DatabaseAgentInternalMessageListeners.iterator();
    boolean test = Listeners.hasNext();
    while(Listeners.hasNext())
    {
        ((PackageDatabaseAgent_ServiceInternalMessagesEventListener)Listeners.next()).ServiceInternalMessagesEventReceived(NewEvent);
    }
    this.incrementSystemLocalTime();
}

//FireServiceDevelopmentAgentInternalMessageQueueEvent
public synchronized void
    AddServiceDevelopmentAgentInternalMessageQueueEventListener(PackageDevelopmentAgent_ServiceInternalMessagesEventListener Listener)
{
    DevelopmentAgentInternalMessageListeners.add(Listener);
}

public synchronized void
    RemoveServiceDevelopmentAgentInternalMessageQueueEventListener(PackageDevelopmentAgent_ServiceInternalMessagesEventListener Listener)
{
    DevelopmentAgentInternalMessageListeners.remove(Listener);
}

public synchronized void

FireServiceDevelopmentAgentInternalMessageQueueEvent()

//creates new event and fires it to be handled
PackageDevelopmentAgent_ServiceInternalMessagesEvent NewEvent =
  new PackageDevelopmentAgent_ServiceInternalMessagesEvent(
    this, this);
Iterator Listeners = DevelopmentAgentInternalMessageListeners.iterator();
boolean test = Listeners.hasNext();
while(Listeners.hasNext())
{
  ((PackageDevelopmentAgent_ServiceInternalMessagesEventListener)Listeners
    .next()).ServiceInternalMessagesEventReceived(NewEvent);
}
this.IncrementSystemLocalTime();

//FireServiceGUIAgentInternalMessageQueueEvent
public synchronized void
  AddServiceGUIAgentInternalMessageQueueEventListener(
    PackageGUIAgent_ServiceInternalMessagesEventListener Listener)
{
  GUIAgentInternalMessageListeners.add(Listener);
}

public synchronized void
  RemoveServiceGUIAgentInternalMessageQueueEventListener(
    PackageGUIAgent_ServiceInternalMessagesEventListener Listener)
{
  GUIAgentInternalMessageListeners.remove(Listener);
}

public synchronized void
  FireServiceGUIAgentInternalMessageQueueEvent()
{
  //creates new event and fires it to be handled
  PackageGUIAgent_ServiceInternalMessagesEvent NewEvent =
    new PackageGUIAgent_ServiceInternalMessagesEvent(
      this, this);
  Iterator Listeners = GUIAgentInternalMessageListeners.iterator();
  boolean test = Listeners.hasNext();
  while(Listeners.hasNext())
  {
    ((PackageGUIAgent_ServiceInternalMessagesEventListener)Listeners
      .next()).ServiceInternalMessagesEventReceived(NewEvent);
  }
  this.IncrementSystemLocalTime();
}
1240 //FireServiceIntelligenceAgentInternalMessageQueueEvent
1241 public synchronized void
1242   AddServiceIntelligenceAgentInternalMessageQueueEventListener(
1243     PackageIntelligenceAgent_ServiceInternalMessagesEventListener Listener)
1244   {
1245     IntelligenceAgentInternalMessageListeners.add(Listener);
1246   }
1247 public synchronized void
1248   RemoveServiceIntelligenceAgentInternalMessageQueueEventListener(
1249     PackageIntelligenceAgent_ServiceInternalMessagesEventListener Listener)
1250   {
1251     IntelligenceAgentInternalMessageListeners.remove(Listener);
1252   }
1253 public synchronized void
1254   FireServiceIntelligenceAgentInternalMessageQueueEvent()
1255   {
1256     //creates new event and fires it to be handled
1257     PackageIntelligenceAgent_ServiceInternalMessagesEvent NewEvent =
1258       new PackageIntelligenceAgent_ServiceInternalMessagesEvent(
1259         this, this);
1260     Iterator Listeners = IntelligenceAgentInternalMessageListeners.iterator();
1261     boolean test = Listeners.hasNext();
1262     while(Listeners.hasNext())
1263       {
1264         ((PackageIntelligenceAgent_ServiceInternalMessagesEventListener)Listeners.next()).ServiceInternalMessagesEventReceived(NewEvent);
1265       }
1266       this.IncrementSystemLocalTime();
1267   }
1268 //FireServiceMessageAnalyzerAgentInternalMessageQueueEvent
1269 public synchronized void
1270   AddServiceMessageAnalyzerAgentInternalMessageQueueEventListener(
1271     PackageMessageAnalyzerAgent_ServiceInternalMessagesEventListener Listener)
1272   {
1273     MessageAnalyzerAgentInternalMessageListeners.add(Listener);
1274   }
1275 public synchronized void
1276   RemoveServiceMessageAnalyzerAgentInternalMessageQueueEventListener(
1277     PackageMessageAnalyzerAgent_ServiceInternalMessagesEventListener Listener)
1278   {
1279     MessageAnalyzerAgentInternalMessageListeners.remove(Listener);
1280   }
1281 public synchronized void
1282   FireServiceMessageAnalyzerAgentInternalMessageQueueEvent()
// creates new event and fires it to be handled
PackageMessageAnalyzerAgent_ServiceInternalMessagesEvent NewEvent =
    new PackageMessageAnalyzerAgent_ServiceInternalMessagesEvent(
        this, this);
Iterator Listeners = MessageAnalyzerAgentInternalMessageListeners.iterator();
boolean test = Listeners.hasNext();
while(Listeners.hasNext()) {((PackageMessageAnalyzerAgent_ServiceInternalMessagesEventListener)Listeners.next()].ServiceInternalMessagesEventReceived(NewEvent);
}
this.IncrementSystemLocalTime();

// FireServiceMessengerAgentInternalMessageQueueEvent
public synchronized void
    AddServiceMessengerAgentInternalMessageQueueEventListener(
        PackageMessengerAgent_ServiceInternalMessagesEventListener Listener) {
        MessengerAgentInternalMessageListeners.add(Listener);
    }

public synchronized void
    RemoveServiceMessengerAgentInternalMessageQueueEventListener(
        PackageMessengerAgent_ServiceInternalMessagesEventListener Listener) {
        MessengerAgentInternalMessageListeners.remove(Listener);
    }

public synchronized void
    FireServiceMessengerAgentInternalMessageQueueEvent() {
    // creates new event and fires it to be handled
    PackageMessageAgent_ServiceInternalMessagesEvent NewEvent =
        new PackageMessageAgent_ServiceInternalMessagesEvent(
            this, this);
    Iterator Listeners = MessengerAgentInternalMessageListeners.iterator();
    boolean test = Listeners.hasNext();
    while(Listeners.hasNext()) {
        ((PackageMessageAgent_ServiceInternalMessagesEventListener)Listeners.next()).ServiceInternalMessagesEventReceived(NewEvent);
    }
    this.IncrementSystemLocalTime();
}

// FireServiceSoftwareTransferAgentInternalMessageQueueEvent
public synchronized void
AddServiceSoftwareTransferAgentInternalMessageQueueEventListener(
    PackageSoftwareTransferAgent_ServiceInternalMessagesEventListener
    Listener)
1339 {
    SoftwareTransferAgentInternalMessageListeners.add(Listener);
1341 }
1342
1343 public synchronized void
RemoveServiceSoftwareTransferAgentInternalMessageQueueEventListener(
    PackageSoftwareTransferAgent_ServiceInternalMessagesEventListener
    Listener)
1345 {
    SoftwareTransferAgentInternalMessageListeners.remove(Listener);
1347 }
1348
1349 public synchronized void
FireServiceSoftwareTransferAgentInternalMessageQueueEvent()
1350 {
    //creates new event and fires it to be handled
    PackageSoftwareTransferAgent_ServiceInternalMessagesEvent
    NewEvent =
    new PackageSoftwareTransferAgent_ServiceInternalMessagesEvent(
        this, this);
    Iterator Listeners =
    SoftwareTransferAgentInternalMessageListeners.iterator();
    boolean test = Listeners.hasNext();
    while(Listeners.hasNext())
    {
        ((PackageSoftwareTransferAgent_ServiceInternalMessagesEventListener)
        Listeners.next()).ServiceInternalMessagesEventReceived(NewEvent);
    }
    this.IncrementSystemLocalTime();
1365 }
1366
1367
1368 //FireServiceSystemsAgentInternalMessageQueueEvent
1369 public synchronized void
AddServiceSystemsAgentInternalMessageQueueEventListener(
    PackageSystemsAgent_ServiceInternalMessagesEventListener
    Listener)
1370 {
    SystemsAgentInternalMessageListeners.add(Listener);
1373 }
1374
1375 public synchronized void
RemoveServiceSystemsAgentInternalMessageQueueEventListener(
    PackageSystemsAgent_ServiceInternalMessagesEventListener
    Listener)
1377 {
    SystemsAgentInternalMessageListeners.remove(Listener);
1379 }
1380
1381 public synchronized void
FireServiceSystemsAgentInternalMessageQueueEvent()
1382 {
    //creates new event and fires it to be handled
    PackageSystemsAgent_ServiceInternalMessagesEvent NewEvent =
new PackageSystemsAgent_ServiceInternalMessagesEvent(
  this, this);
Iterator Listeners = SystemsAgentInternalMessageListeners.iterator();
boolean test = Listeners.hasNext();
while(Listeners.hasNext())
  {
    (PackageSystemsAgent_ServiceInternalMessagesEventListener) Listeners.next().ServiceInternalMessagesEventReceived(NewEvent);
  }
this.IncrementSystemLocalTime();
}

//FireServiceVersionTrackingAgentInternalMessageQueueEvent
public synchronized void AddServiceVersionTrackingAgentInternalMessageQueueEventListener(
  PackageVersionTrackingAgent_ServiceInternalMessagesEventListener Listener)
{
  VersionTrackingAgentInternalMessageListeners.add(Listener);
}

public synchronized void RemoveServiceVersionTrackingAgentInternalMessageQueueEventListener(
  PackageVersionTrackingAgent_ServiceInternalMessagesEventListener Listener)
{
  VersionTrackingAgentInternalMessageListeners.remove(Listener);
}

public synchronized void FireServiceVersionTrackingAgentInternalMessageQueueEvent()
{
  //creates new event and fires it to be handled
  PackageVersionTrackingAgent_ServiceInternalMessagesEvent NewEvent =
    new PackageVersionTrackingAgent_ServiceInternalMessagesEvent(
      this, this);
Iterator Listeners = VersionTrackingAgentInternalMessageListeners.iterator();
while(Listeners.hasNext())
  {
    ((PackageVersionTrackingAgent_ServiceInternalMessagesEventListener) Listeners.next()).ServiceInternalMessagesEventReceived(NewEvent);
  }
this.IncrementSystemLocalTime();
}

//FireServiceVirtualNetworksManagerAgentInternalMessageQueueEvent
public synchronized void AddServiceVirtualNetworksManagerAgentInternalMessageQueueEventListener(PackageVirtualNetworksManagerAgent_ServiceInternalMessagesEventListener Listener) {
    VirtualNetworksManagerAgentInternalMessageListeners.add(Listener);
}

public synchronized void RemoveServiceVirtualNetworksManagerAgentInternalMessageQueueEventListener(PackageVirtualNetworksManagerAgent_ServiceInternalMessagesEventListener Listener) {
    VirtualNetworksManagerAgentInternalMessageListeners.remove(Listener);
}

public synchronized void FireServiceVirtualNetworksManagerAgentInternalMessageQueueEvent() {
    //creates new event and fires it to be handled
    PackageVirtualNetworksManagerAgent_ServiceInternalMessagesEvent NewEvent =
        new PackageVirtualNetworksManagerAgent_ServiceInternalMessagesEvent(this, this);

    Iterator Listeners = VirtualNetworksManagerAgentInternalMessageListeners.iterator();
    boolean test = Listeners.hasNext();
    while(Listeners.hasNext()) {
        ((PackageVirtualNetworksManagerAgent_ServiceInternalMessagesEventListener) Listeners.next()).ServiceInternalMessagesEventReceived(NewEvent);
    }

    this.IncrementSystemLocalTime();
}

public synchronized void AddServiceColinAgentInternalMessageQueueEventListener(PackageColinAgent_ServiceInternalMessagesEventListener Listener) {
    ColinAgentInternalMessageListeners.add(Listener);
}

public synchronized void RemoveServiceColinAgentInternalMessageQueueEventListener(PackageColinAgent_ServiceInternalMessagesEventListener Listener) {
    ColinAgentInternalMessageListeners.remove(Listener);
}

public synchronized void FireServiceColinAgentInternalMessageQueueEvent() {
    //creates new event and fires it to be handled
    PackageColinAgent_ServiceInternalMessagesEvent NewEvent =
        new PackageColinAgent_ServiceInternalMessagesEvent(this, this);

    Iterator Listeners = ColinAgentInternalMessageListeners.iterator();
    boolean test = Listeners.hasNext();
    while(Listeners.hasNext()) {
        ((PackageColinAgent_ServiceInternalMessagesEventListener) Listeners.next()).ServiceInternalMessagesEventReceived(NewEvent);
    }

    this.IncrementSystemLocalTime();
}
1482 PackageColinAgent_ServiceInternalMessagesEvent NewEvent =
1483 new PackageColinAgent_ServiceInternalMessagesEvent(
1484 this, this);
1485 Iterator Listeners = ColinAgentInternalMessageListeners.iterator();
1486 boolean test = Listeners.hasNext();
1487 while(Listeners.hasNext())
1488 {
1489   ((PackageColinAgent_ServiceInternalMessagesEventListener)Listeners.next(
1490       )).ServiceInternalMessagesEventReceived(NewEvent);
1491 }
1492 this.IncrementSystemLocalTime();
1493 }
1494 //end of event methods
1495
1496 final HolonicTechnologyPlatformMaster SystemMaster = this;
1497
1498 /************ /
1499 /* main function */
1500 /************ /
1501
1502 public void RunHTP()
1503 {
1504   final HolonicTechnologyPlatformMaster SystemMaster = this;
1505   /************ /
1506   /* create and add listeners */
1507   /************ /
1508   //internal messaging function listeners
1509   PackageExecutable_SwitchMechanismHandleInternalMessagesEventListener
1510   HandleInternalMessagesEventListerner = new
1511   PackageExecutable_SwitchMechanismHandleInternalMessagesEventHandler();
1512   SystemMaster.AddHandleInternalMessagesEventListener(HandleInternalMessagesEventListener);
1513   PackageAdministrationAgent_ServiceInternalMessagesEventListener
1514   ServiceAdministrationAgentInternalMessagesEventListener = new
1515   PackageAdministrationAgent_ServiceInternalMessagesEventHandler();
1516   SystemMaster.AddServiceAdministrationAgentInternalMessageQueueEventListener(
1517       ServiceAdministrationAgentInternalMessagesEventListener);
1518   PackageAlgorithmsAgent_ServiceInternalMessagesEventListener
1519   ServiceAlgorithmsAgentInternalMessagesEventListener = new
1520   PackageAlgorithmsAgent_ServiceInternalMessagesEventHandler();
1521   SystemMaster.AddServiceAlgorithmsAgentInternalMessageQueueEventListener(
1522       ServiceAlgorithmsAgentInternalMessagesEventListener);
1523   PackageApplicationsManagerAgent_ServiceInternalMessagesEventListener
1524   ServiceApplicationsManagerAgentInternalMessagesEventListener = new
1525   PackageApplicationsManagerAgent_ServiceInternalMessagesEventHandler();
1526   /************ /
1527 SystemMaster.AddServiceApplicationsManagerAgentInternalMessageQueueEventListener(ServiceApplicationsManagerAgentInternalMessagesEventListener);
1528 PackageConsoleAgent_ServiceInternalMessagesEventListener
ServiceConsoleAgentInternalMessagesEventListener = new PackageConsoleAgent_ServiceInternalMessagesEventHandler();
1530 SystemMaster.AddServiceConsoleAgentInternalMessageQueueEventListener(ServiceConsoleAgentInternalMessagesEventListener);
1532 PackageDatabaseAgent_ServiceInternalMessagesEventListener
ServiceDatabaseAgentInternalMessagesEventListener = new PackageDatabaseAgent_ServiceInternalMessagesEventHandler();
1533 SystemMaster.AddServiceDatabaseAgentInternalMessageQueueEventListener(ServiceDatabaseAgentInternalMessagesEventListener);
1535 PackageDevelopmentAgent_ServiceInternalMessagesEventListener
ServiceDevelopmentAgentInternalMessagesEventListener = new PackageDevelopmentAgent_ServiceInternalMessagesEventHandler();
1536 SystemMaster.AddServiceDevelopmentAgentInternalMessageQueueEventListener(ServiceDevelopmentAgentInternalMessagesEventListener);
1538 PackageGUIAgent_ServiceInternalMessagesEventListener
ServiceGUIAgentInternalMessagesEventListener = new PackageGUIAgent_ServiceInternalMessagesEventHandler();
1539 SystemMaster.AddServiceGUIAgentInternalMessageQueueEventListener(ServiceGUIAgentInternalMessagesEventListener);
1541 PackageIntelligenceAgent_ServiceInternalMessagesEventListener
ServiceIntelligenceAgentInternalMessagesEventListener = new PackageIntelligenceAgent_ServiceInternalMessagesEventHandler();
1542 SystemMaster.AddServiceIntelligenceAgentInternalMessageQueueEventListener(ServiceIntelligenceAgentInternalMessagesEventListener);
1544 PackageMessageAnalyzerAgent_ServiceInternalMessagesEventListener
ServiceMessageAnalyzerAgentInternalMessagesEventListener = new PackageMessageAnalyzerAgent_ServiceInternalMessagesEventHandler();
1545 SystemMaster.AddServiceMessageAnalyzerAgentInternalMessageQueueEventListener(ServiceMessageAnalyzerAgentInternalMessagesEventListener);
1547 PackageMessengerAgent_ServiceInternalMessagesEventListener
ServiceMessengerAgentInternalMessagesEventListener = new PackageMessengerAgent_ServiceInternalMessagesEventHandler();
1548 SystemMaster.AddServiceMessengerAgentInternalMessageQueueEventListener(ServiceMessengerAgentInternalMessagesEventListener);
1550 PackageSoftwareTransferAgent_ServiceInternalMessagesEventListener
ServiceSoftwareTransferAgentInternalMessagesEventListener = new PackageSoftwareTransferAgent_ServiceInternalMessagesEventHandler();
1551 SystemMaster.AddServiceSoftwareTransferAgentInternalMessageQueueEventListener(ServiceSoftwareTransferAgentInternalMessagesEventListener);
1553 PackageSystemsAgent_ServiceInternalMessagesEventListener
ServiceSystemsAgentInternalMessagesEventListener = new PackageSystemsAgent_ServiceInternalMessagesEventHandler();
1554 SystemMaster.AddServiceSystemsAgentInternalMessageQueueEventListener(ServiceSystemsAgentInternalMessagesEventListener);
1555    PackageVersionTrackingAgent_ServiceInternalMessagesEventListener
1556    ServiceVersionTrackingAgentInternalMessagesEventListener = new
1557    PackageVersionTrackingAgent_ServiceInternalMessagesEventListener();
1558    SystemMaster.AddServiceVersionTrackingAgentInternalMessageQueueEventListener(ServiceVersionTrackingAgentInternalMessagesEventListener);
1559    PackageVirtualNetworksManagerAgent_ServiceInternalMessagesEventListener
1560    ServiceVirtualNetworksManagerAgentInternalMessagesEventListener = new
1561    PackageVirtualNetworksManagerAgent_ServiceInternalMessagesEventListener();
1562    SystemMaster.AddServiceVirtualNetworksManagerAgentInternalMessageQueueEventListener(ServiceVirtualNetworksManagerAgentInternalMessagesEventListener);
1563    PackageCabinetAgent_ServiceInternalMessagesEventListener
1564    ServiceCabinetAgentInternalMessagesEventListener = new
1565    PackageCabinetAgent_ServiceInternalMessagesEventListener();
1566    SystemMaster.AddServiceCabinetAgentInternalMessageQueueEventListener(ServiceCabinetAgentInternalMessagesEventListener);
1567    PackageAlgorithmsAgent_CreateNewAlgorithmVariableEventListener
1568    CreateNewAlgorithmVariableEventListener = new
1569    PackageAlgorithmsAgent_CreateNewAlgorithmVariableEventListener();
1570    SystemMaster.AddCreateNewAlgorithmVariableEventListener(CreateNewAlgorithmVariableEventListener);
1571    PackageAlgorithmsAgent_DeleteAlgorithmVariableEventListener
1572    DeleteAlgorithmVariableEventListener = new
1573    PackageAlgorithmsAgent_DeleteAlgorithmVariableEventListener();
1574    SystemMaster.AddDeleteAlgorithmVariableEventListener(DeleteAlgorithmVariableEventListener);
1575    PackageAlgorithmsAgent_ModifyAlgorithmVariableEventListener
1576    ModifyAlgorithmVariableEventListener = new
1577    PackageAlgorithmsAgent_ModifyAlgorithmVariableEventListener();
1578    SystemMaster.AddModifyAlgorithmVariableEventListener(ModifyAlgorithmVariableEventListener);
1579    PackageAlgorithmsAgent_ModifyAlgorithmListEventListener
1580    ModifyAlgorithmListEventListener = new
1581    PackageAlgorithmsAgent_ModifyAlgorithmListEventListener();
1582    SystemMaster.AddModifyAlgorithmListEventListener(ModifyAlgorithmListEventListener);
1583    PackageAlgorithmsAgent_InitiateAlgorithmEventListener
1584    InitiateAlgorithmEventListener = new
1585    PackageAlgorithmsAgent_InitiateAlgorithmEventListener();
1586    SystemMaster.AddInitiateAlgorithmEventListener(InitiateAlgorithmEventListener);
1587    PackageAlgorithmsAgent_InitiateAlgorithmEventListener
1588    //administration agent function listeners
1589
1585 PackageAdministrationAgent_InitiateSaveAndQuitEventListener
   InitiateSaveAndQuitEventListener = new
   PackageAdministrationAgent_InitiateSaveAndQuitEventHandler();
1586 SystemMaster.AddInitiateSaveAndQuitEventListener(InitiateSaveAndQuitEvent-
   Listener);
1587
1588 //database agent function listeners
1589 PackageDatabaseAgent_BackupHolonicSystemNowEventListener
   BackupHolonicSystemNowEventListener = new
   PackageDatabaseAgent_BackupHolonicSystemNowEventHandler();
1590 SystemMaster.AddBackupHolonicSystemNowEventListener(BackupHolonicSystemN-
   owEventListener);
1591
1592 PackageDatabaseAgent_RestoreHolonicSystemNowEventListener
   RestoreHolonicSystemNowEventListener = new
   PackageDatabaseAgent_RestoreHolonicSystemNowEventHandler();
1593 SystemMaster.AddRestoreHolonicSystemNowEventListener(RestoreHolonicSyste-
   mNowEventListener);
1594
1595 //applications manager agent function listeners
1596 PackageApplicationsManagerAgent_RegisterNewApplicationEventListener
   RegisterNewApplicationEventListener = new
   PackageApplicationsManagerAgent_RegisterNewApplicationEventHandler();
1597 SystemMaster.AddRegisterNewApplicationEventListener(RegisterNewApplicati-
   onEventListener);
1598
1599 PackageApplicationsManagerAgent_UpdateExistingApplicationEventListener
   UpdateExistingApplicationEventListener = new
   PackageApplicationsManagerAgent_UpdateExistingApplicationEventHandler();
1600 SystemMaster.AddUpdateExistingApplicationEventListener(UpdateExistingAp-
   plicationEventListener);
1601
1602 PackageApplicationsManagerAgent_DeleteExistingApplicationEventListener
   DeleteExistingApplicationEventListener = new
   PackageApplicationsManagerAgent_DeleteExistingApplicationEventHandler();
1603 SystemMaster.AddDeleteExistingApplicationEventListener(DeleteExistingApp-
   plicationEventListener);
1604
1605 PackageApplicationsManagerAgent_RegisterNewApplicationServiceEventListener
   RegisterNewApplicationServiceEventListener = new
   PackageApplicationsManagerAgent_RegisterNewApplicationServiceEventHandler();
1606 SystemMaster.AddRegisterNewApplicationServiceEventListener(RegisterNewAp-
   plicationServiceEventListener);
1607
1608 PackageApplicationsManagerAgent_UpdateExistingApplicationServiceEventListener
   UpdateExistingApplicationServiceEventListener = new
   PackageApplicationsManagerAgent_UpdateExistingApplicationServiceEventHandler();
1609 SystemMaster.AddUpdateExistingApplicationServiceEventListener(UpdateEx-
   istingApplicationServiceEventListener);
1610
1611 PackageApplicationsManagerAgent_DeleteExistingApplicationServiceEventListener
   DeleteExistingApplicationServiceEventListener = new
   PackageApplicationsManagerAgent_DeleteExistingApplicationServiceEventListener();
1612 SystemMaster.AddDeleteExistingApplicationServiceEventListener(DeleteExis-
   tingApplicationServiceEventListener);
PackageApplicationsManagerAgent_DeleteExistingApplicationServiceEventHandler();

1614 SystemMaster.AddDeleteExistingApplicationServiceEventListener(DeleteExistingApplicationServiceEventListener);

1615 PackageApplicationsManagerAgent_SendAvailableNodesListToIntelligenceLayerEventListener SendAvailableNodesListToIntelligenceLayerEventListener = new PackageApplicationsManagerAgent_SendAvailableNodesListToIntelligenceLayerEventHandler();

1616 SystemMaster.AddSendAvailableNodesListToIntelligenceLayerEventListener(SendAvailableNodesListToIntelligenceLayerEventListener);

1617 SystemMaster.AddSendAvailableNodesListToIntelligenceLayerEventListener(SendAvailableNodesListToIntelligenceLayerEventListener);

1618 PackageApplicationsManagerAgent_InquireAboutApplicationServicesEventListener InquireAboutApplicationServicesEventListener = new PackageApplicationsManagerAgent_InquireAboutApplicationServicesEventListener();

1619 SystemMaster.AddInquireAboutApplicationServicesEventListener(InquireAboutApplicationServicesEventListener);

1620 SystemMaster.AddInquireAboutApplicationServicesEventListener(InquireAboutApplicationServicesEventListener);

1621 PackageApplicationsManagerAgent_SearchForApplicationServiceEventListener SearchForApplicationServiceEventListener = new PackageApplicationsManagerAgent_SearchForApplicationServiceEventListener();

1622 SystemMaster.AddSearchForApplicationServiceEventListener(SearchForApplicationServiceEventListener);

1623 SystemMaster.AddSearchForApplicationServiceEventListener(SearchForApplicationServiceEventListener);

1624 PackageApplicationsManagerAgent_SendNodeWithFoundApplicationServiceToIntelligenceLayerEventListener SendNodeWithFoundApplicationServiceToIntelligenceLayerEventListener = New PackageApplicationsManagerAgent_SendNodeWithFoundApplicationServiceToIntelligenceLayerEventHandler();

1625 SystemMaster.AddSendNodeWithFoundApplicationServiceToIntelligenceLayerEventListener(SendNodeWithFoundApplicationServiceToIntelligenceLayerEventListener);

1626 SystemMaster.AddSendNodeWithFoundApplicationServiceToIntelligenceLayerEventListener(SendNodeWithFoundApplicationServiceToIntelligenceLayerEventListener);

1627 PackageApplicationsManagerAgent_SendIntelligenceToIntelligenceEventListener SendIntelligenceToIntelligenceEventListener = new PackageApplicationsManagerAgent_SendIntelligenceToIntelligenceEventHandler();

1628 SystemMaster.AddSendIntelligenceToIntelligenceEventListener(SendIntelligenceToIntelligenceEventListener);

1629 SystemMaster.AddSendIntelligenceToIntelligenceEventListener(SendIntelligenceToIntelligenceEventListener);

1630 PackageApplicationsManagerAgent_SendHolonicsToIntelligenceEventListener SendHolonicsToIntelligenceEventListener = new PackageApplicationsManagerAgent_SendHolonicsToIntelligenceEventHandler();

1631 SystemMaster.AddSendHolonicsToIntelligenceEventListener(SendHolonicsToIntelligenceEventListener);

1632 SystemMaster.AddSendHolonicsToIntelligenceEventListener(SendHolonicsToIntelligenceEventListener);

1633 PackageApplicationsManagerAgent_SendListOfVirtualNetworksEventListener SendListOfVirtualNetworksEventListener = new PackageApplicationsManagerAgent_SendListOfVirtualNetworksEventHandler();

1634 SystemMaster.AddSendListOfVirtualNetworksEventListener(SendListOfVirtualNetworksEventListener);

1635 SystemMaster.AddSendListOfVirtualNetworksEventListener(SendListOfVirtualNetworksEventListener);
1636  PackageApplicationsManagerAgent_BackupQueueForSendingSplitUpMessagesTimerUpFunctionEventListener
1637      BackupQueueForSendingSplitUpMessagesTimerUpFunctionEventListener =
1638          new PackageApplicationsManagerAgent_BackupQueueForSendingSplitUpMessagesTimerUpFunctionEventListener();
1639  SystemMaster.AddBackupQueueForSendingSplitUpMessagesTimerUpFunctionEventListener(BackupQueueForSendingSplitUpMessagesTimerUpFunctionEventListener);
1640  PackageApplicationsManagerAgent_DeleteBackupSplitUpMessageEventListener
1641      DeleteBackupSplitUpMessageEventListener = new PackageApplicationsManagerAgent_DeleteBackupSplitUpMessageEventHandler();
1642  SystemMaster.AddDeleteBackupSplitUpMessageEventListener(DeleteBackupSplitUpMessageEventListener);
1643  PackageApplicationsManagerAgent_SendIntelligenceToIntelligenceViaHolonicEventListener
1644      SendIntelligenceToIntelligenceViaHolonicEventListener = new PackageApplicationsManagerAgent_SendIntelligenceToIntelligenceViaHolonicEventListener();
1645  SystemMaster.AddSendIntelligenceToIntelligenceViaHolonicEventListener(SendIntelligenceToIntelligenceViaHolonicEventListener);
1646  PackageApplicationsManagerAgent_SendHolonicToIntelligenceViaHolonicEventListener
1647      SendHolonicToIntelligenceViaHolonicEventListener = new PackageApplicationsManagerAgent_SendHolonicToIntelligenceViaHolonicEventListener();
1648  SystemMaster.AddSendHolonicToIntelligenceViaHolonicEventListener(SendHolonicToIntelligenceViaHolonicEventListener);
1650  //console listeners
1651  PackageConsoleAgent_AdministrationConsoleEventListener
1652      AdministrationConsoleEventListener = new PackageConsoleAgent_AdministrationConsoleEventHandler();
1653  SystemMaster.AddAdministrationConsoleEventListener(AdministrationConsoleEventListener);
1654  PackageConsoleAgent_VirtualNetworksManagerConsoleEventListener
1655      VirtualNetworksManagerConsoleEventListener = new PackageConsoleAgent_VirtualNetworksManagerConsoleEventHandler();
1656  SystemMaster.AddVirtualNetworksManagerConsoleEventListener(VirtualNetworksManagerConsoleEventListener);
1657  PackageConsoleAgent_ApplicationsManagerConsoleEventListener
1658      ApplicationsManagerConsoleEventListener = new PackageConsoleAgent_ApplicationsManagerConsoleEventHandler();
1659  SystemMaster.AddApplicationsManagerConsoleEventListener(ApplicationsManagerConsoleEventListener);
1660  PackageConsoleAgent_SystemsConsoleEventListener
1661      SystemsConsoleEventListener = new PackageConsoleAgent_SystemsConsoleEventHandler();
SystemMaster.AddSystemsConsoleEventListener(SystemsConsoleEventListener);

PackageConsoleAgent_OutputToConsoleEventListener = new PackageConsoleAgent_OutputToConsoleEventHandler();
SystemMaster.AddOutputToConsoleEventListener(OutputToConsoleEventListener);

PackageConsoleAgent_SoftwareTransferConsoleEventListener = new PackageConsoleAgent_SoftwareTransferConsoleEventHandler();
SystemMaster.AddSoftwareTransferConsoleEventListener(SoftwareTransferConsoleEventListener);

PackageConsoleAgent_DevelopmentConsoleEventListener = new PackageConsoleAgent_DevelopmentConsoleEventHandler();
SystemMaster.AddDevelopmentConsoleEventListener(DevelopmentConsoleEventListener);

PackageConsoleAgent_VersionTrackingConsoleEventListener = new PackageConsoleAgent_VersionTrackingConsoleEventHandler();
SystemMaster.AddVersionTrackingConsoleEventListener(VersionTrackingConsoleEventListener);

PackageConsoleAgent_DatabaseConsoleEventListener = new PackageConsoleAgent_DatabaseConsoleEventHandler();
SystemMaster.AddDatabaseConsoleEventListener(DatabaseConsoleEventListener);

PackageConsoleAgent_AlgorithmsConsoleEventListener = new PackageConsoleAgent_AlgorithmsConsoleEventHandler();
SystemMaster.AddAlgorithmsConsoleEventListener(AlgorithmsConsoleEventListener);

PackageConsoleAgent_ShowAgentMenusEventListener = new PackageConsoleAgent_ShowAgentMenusEventHandler();
SystemMaster.AddShowAgentMenusEventListener(ShowAgentMenusEventListener);

//software transfer listeners
PackageSoftwareTransferAgent_SendByteCodeEventListener = new PackageSoftwareTransferAgent_SendByteCodeEventHandler();
SystemMaster.AddSendByteCodeEventListener(SendByteCodeEventListener);

PackageSoftwareTransferAgent_AddSegmentOfByteCodeEventListener = new PackageSoftwareTransferAgent_AddSegmentOfByteCodeEventHandler();
SystemMaster.AddAddSegmentOfByteCodeEventListener(AddSegmentOfByteCodeEventListener);

PackageSoftwareTransferAgent_ConcludeSoftwareTransferEventListener = new PackageSoftwareTransferAgent_ConcludeSoftwareTransferEventHandler();
SystemMaster.AddConcludeSoftwareTransferEventListener(ConcludeSoftwareTransferEventListener);
PackageSoftwareTransferAgent_ConcludeSoftwareTransferEventHandler();

1693 SystemMaster.AddConcludeSoftwareTransferEventListener(ConcludeSoftwareTransferEventListener);

1694 PackageSoftwareTransferAgent_InitiateSoftwareUpdateEventListener
InitiateSoftwareUpdateEventListener = new PackageSoftwareTransferAgent_InitiateSoftwareUpdateEventHandler();

1695 SystemMaster.AddInitiateSoftwareUpdateEventListener(InitiateSoftwareUpdateEventListener);

1696 PackageSoftwareTransferAgent_TestByteCodeEventListener
TestByteCodeEventListener = new PackageSoftwareTransferAgent_TestByteCodeEventHandler();

1697 SystemMaster.AddTestByteCodeEventListener(TestByteCodeEventListener);

1698 PackageSoftwareTransferAgent_CopyAndRenameClassFileEventListener
CopyAndRenameClassFileEventListener = new PackageSoftwareTransferAgent_CopyAndRenameClassFileEventHandler();

1699 SystemMaster.AddCopyAndRenameClassFileEventListener(CopyAndRenameClassFileEventListener);

1700 PackageSoftwareTransferAgent_SendApplicationSoftwareEventListener
SendApplicationSoftwareEventListener = new PackageSoftwareTransferAgent_SendApplicationSoftwareEventHandler();

1701 SystemMaster.AddSendApplicationSoftwareEventListener(SendApplicationSoftwareEventListener);

1702 PackageSoftwareTransferAgent_SendAgentSoftwareEventListener
SendAgentSoftwareEventListener = new PackageSoftwareTransferAgent_SendAgentSoftwareEventHandler();

1703 SystemMaster.AddSendAgentSoftwareEventListener(SendAgentSoftwareEventListener);

1704 PackageIntelligenceAgent_SendExternalMessageForIntelligenceAnalysisEventListener
SendExternalMessageForIntelligenceAnalysisEventListener = new PackageIntelligenceAgent_SendExternalMessageForIntelligenceAnalysisEventHandler();

1705 SystemMaster.AddSendExternalMessageForIntelligenceAnalysisEventListener(SendExternalMessageForIntelligenceAnalysisEventListener);

1706 PackageDevelopmentAgent_CreateNewAgentCodeUsingTemplateEventListener
CreateNewAgentCodeUsingTemplateEventListener = new PackageDevelopmentAgent_CreateNewAgentCodeUsingTemplateEventHandler();

1707 SystemMaster.AddCreateNewAgentCodeUsingTemplateEventListener(CreateNewAgentCodeUsingTemplateEventListener);

1708 PackageDevelopmentAgent_CreateNewApplicationAgentCodeUsingTemplateEventListener

1709 SystemMaster.AddCreateNewApplicationAgentCodeUsingTemplateEventListener(CreateNewApplicationAgentCodeUsingTemplateEventListener);
CreateNewApplicationAgentCodeUsingTemplateEventListener();

1722 PackageDevelopmentAgent_ListAgentWithSystemEventListener
1723 ListAgentWithSystemEventListener = new
1724 PackageDevelopmentAgent_ListAgentWithSystemEventHandler();
1725 SystemMaster.AddListAgentWithSystemEventListener(ListAgentWithSystemEventListener);
1726
1727 //message analyzer listeners
1728 PackageMessageAnalyzerAgent_MessageAnalysisEventListener
1729 MessageAnalysisEventListener = new
1730 PackageMessageAnalyzerAgent_MessageAnalysisEventHandler();
1731 SystemMaster.AddMessageAnalysisEventListener(MessageAnalysisEventListener);
1732
1733 //messenger listeners (intelligence specific)
1734 PackageMessengerAgent_SendIntelligenceResultsToMessengerAgentEventListener
1735 SendIntelligenceResultsToMessengerAgentEventListener = new
1736 PackageMessengerAgent_SendIntelligenceResultsToMessengerAgentEventHandler();
1737 SystemMaster.AddSendIntelligenceResultsToMessengerAgentEventListener(SendIntelligenceResultsToMessengerAgentEventListener);
1738
1739 //input listeners
1740 PackageMessengerAgent_MainInputQueueEventListener
1741 MainInputQueueEventListener = new
1742 PackageMessengerAgent_MainInputQueueEventHandler();
1743 SystemMaster.AddMainInputQueueEventListener(MainInputQueueEventListener);
1744 PackageMessengerAgent_ApplicationsInputQueueEventListener
1745 ApplicationsInputQueueEventListener = new
1746 PackageMessengerAgent_ApplicationsInputQueueEventHandler();
1747 SystemMaster.AddApplicationsInputQueueEventListener(ApplicationsInputQueueEventListener);
1748 PackageMessengerAgent_AdminApplicationsInputQueueEventListener
1749 AdminApplicationsInputQueueEventListener = new
1750 PackageMessengerAgent_AdminApplicationsInputQueueEventHandler();
1751 SystemMaster.AddAdminApplicationsInputQueueEventListener(AdminApplicationsInputQueueEventListener);
1752 PackageMessengerAgent_UserApplicationsInputQueueEventListener
1753 UserApplicationsInputQueueEventListener = new
1754 PackageMessengerAgent_UserApplicationsInputQueueEventHandler();
1755 SystemMaster.AddUserApplicationsInputQueueEventListener(UserApplicationsInputQueueEventListener);
1756 PackageMessengerAgent_VirtualNetworksInputQueueEventListener
1757 VirtualNetworksInputQueueEventListener = new
1758 PackageMessengerAgent_VirtualNetworksInputQueueEventHandler();
1759 SystemMaster.AddVirtualNetworksInputQueueEventListener(VirtualNetworksInputQueueEventListener);
1760 PackageMessengerAgent_AdminVirtualNetworksInputQueueEventListener
AdminVirtualNetworksInputQueueEventListener = new PackageMessengerAgent_AdminVirtualNetworksInputQueueEventHandler();
SystemMaster.AddAdminVirtualNetworksInputQueueEventListener(AdminVirtualNetworksInputQueueEventListener);

PackageMessengerAgent_UserVirtualNetworksInputQueueEventListener
UserVirtualNetworksInputQueueEventListener = new PackageMessengerAgent_UserVirtualNetworksInputQueueEventHandler();
SystemMaster.AddUserVirtualNetworksInputQueueEventListener(UserVirtualNetworksInputQueueEventListener);

// output listeners
PackageMessengerAgent_MainOutputQueueEventListener
MainOutputQueueEventListener = new PackageMessengerAgent_MainOutputQueueEventHandler();
SystemMaster.AddMainOutputQueueEventListener(MainOutputQueueEventListener);

PackageMessengerAgent_BackupMainOutputQueueEventListener
BackupMainOutputQueueEventListener = new PackageMessengerAgent_BackupMainOutputQueueEventHandler();
SystemMaster.AddBackupMainOutputQueueEventListener(BackupMainOutputQueueEventListener);

PackageMessengerAgent_ApplicationsOutputQueueEventListener
ApplicationsOutputQueueEventListener = new PackageMessengerAgent_ApplicationsOutputQueueEventHandler();
SystemMaster.AddApplicationsOutputQueueEventListener(ApplicationsOutputQueueEventListener);

PackageMessengerAgent_AdminApplicationsOutputQueueEventListener
AdminApplicationsOutputQueueEventListener = new PackageMessengerAgent_AdminApplicationsOutputQueueEventHandler();
SystemMaster.AddAdminApplicationsOutputQueueEventListener(AdminApplicationsOutputQueueEventListener);

PackageMessengerAgent_UserApplicationsOutputQueueEventListener
UserApplicationsOutputQueueEventListener = new PackageMessengerAgent_UserApplicationsOutputQueueEventHandler();
SystemMaster.AddUserApplicationsOutputQueueEventListener(UserApplicationsOutputQueueEventListener);

PackageMessengerAgent_VirtualNetworksOutputQueueEventListener
VirtualNetworksOutputQueueEventListener = new PackageMessengerAgent_VirtualNetworksOutputQueueEventHandler();
SystemMaster.AddVirtualNetworksOutputQueueEventListener(VirtualNetworksOutputQueueEventListener);

PackageMessengerAgent_AdminVirtualNetworksOutputQueueEventListener
AdminVirtualNetworksOutputQueueEventListener = new PackageMessengerAgent_AdminVirtualNetworksOutputQueueEventHandler();
SystemMaster.AddAdminVirtualNetworksOutputQueueEventListener(AdminVirtualNetworksOutputQueueEventListener);

PackageMessengerAgent_UserVirtualNetworksOutputQueueEventListener
UserVirtualNetworksOutputQueueEventListener = new PackageMessengerAgent_UserVirtualNetworksOutputQueueEventHandler();
SystemMaster.AddUserVirtualNetworksOutputQueueEventListener(UserVirtualNetworksOutputQueueEventListener);

306
PackageMessengerAgent_UserVirtualNetworksOutputQueueEventHandler();

SystemMaster.AddUserVirtualNetworksOutputQueueEventListener(UserVirtualNetworksOutputQueueEventListener);

//system function listeners
PackageSystemsAgent_ReadClassLoadingEventListener
ReadClassLoadingEventListener = new
PackageSystemsAgent_ReadClassLoadingEventListener();
SystemMaster.AddReadClassLoadingEventListener(ReadClassLoadingEventListener);

PackageSystemsAgent_ReadCompilationSystemEventListener
ReadCompilationSystemEventListener = new
PackageSystemsAgent_ReadCompilationSystemEventListener();
SystemMaster.AddReadCompilationSystemEventListener(ReadCompilationSystemEventListener);

PackageSystemsAgent_ReadMemorySystemEventListener
ReadMemorySystemEventListener = new
PackageSystemsAgent_ReadMemorySystemEventListener();
SystemMaster.AddReadMemorySystemEventListener(ReadMemorySystemEventListener);

PackageSystemsAgent_ReadMemoryManagerEventListener
ReadMemoryManagerEventListener = new
PackageSystemsAgent_ReadMemoryManagerEventListener();
SystemMaster.AddReadMemoryManagerEventListener(ReadMemoryManagerEventListener);

PackageSystemsAgent_ReadMemoryPoolEventListener
ReadMemoryPoolEventListener = new
PackageSystemsAgent_ReadMemoryPoolEventListener();
SystemMaster.AddReadMemoryPoolEventListener(ReadMemoryPoolEventListener);

PackageSystemsAgent_ReadGarbageCollectorEventListener
ReadGarbageCollectorEventListener = new
PackageSystemsAgent_ReadGarbageCollectorEventListener();
SystemMaster.AddReadGarbageCollectorEventListener(ReadGarbageCollectorEventListener);

PackageSystemsAgent_ReadThreadingSystemEventListener
ReadThreadingSystemEventListener = new
PackageSystemsAgent_ReadThreadingSystemEventListener();
SystemMaster.AddReadThreadingSystemEventListener(ReadThreadingSystemEventListener);

PackageSystemsAgent_ReadRuntimeSystemEventListener
ReadRuntimeSystemEventListener = new
PackageSystemsAgent_ReadRuntimeSystemEventListener();
SystemMaster.AddReadRuntimeSystemEventListener(ReadRuntimeSystemEventListener);

PackageSystemsAgent_ReadOperatingSystemEventListener
ReadOperatingSystemEventListener = new
PackageSystemsAgent_ReadOperatingSystemEventHandler();
1814 SystemMaster.AddReadOperatingSystemEventListener(ReadOperatingSystemEventListener);
1815
1816 //virtual networks manager agent function listeners
1817 PackageVirtualNetworksManagerAgent_CreateNewVirtualNetworkEventListener
CreateNewVirtualNetworkEventListener = new
PackageVirtualNetworksManagerAgent_CreateNewVirtualNetworkEventHandler();
1819 SystemMaster.AddCreateNewVirtualNetworkEventListener(CreateNewVirtualNetworkEventListener);
1820
1821 PackageVirtualNetworksManagerAgent_DeleteExistingVirtualNetworkEventListener
DeleteExistingVirtualNetworkEventListener = new
PackageVirtualNetworksManagerAgent_DeleteExistingVirtualNetworkEventHandler();
1822 SystemMaster.AddDeleteExistingVirtualNetworkEventListener(DeleteExistingVirtualNetworkEventListener);
1823
1824 PackageVirtualNetworksManagerAgent_CreateNewAvailableNodeEventListener
CreateNewAvailableNodeEventListener = new
PackageVirtualNetworksManagerAgent_CreateNewAvailableNodeEventHandler();
1825 SystemMaster.AddCreateNewAvailableNodeEventListener(CreateNewAvailableNodeEventListener);
1826
1827 PackageVirtualNetworksManagerAgent_RegisterNodeOnNetworkEventListener
RegisterNodeOnNetworkEventListener = new
PackageVirtualNetworksManagerAgent_RegisterNodeOnNetworkEventHandler();
1828 SystemMaster.AddRegisterNodeOnNetworkEventListener(RegisterNodeOnNetworkEventListener);
1829
1830 PackageVirtualNetworksManagerAgent_InformNewNodeOfOtherAvailableNodesEventListener
InformNewNodeOfOtherAvailableNodesEventListener = new
PackageVirtualNetworksManagerAgent_InformNewNodeOfOtherAvailableNodesEventHandler();
1831 SystemMaster.AddInformNewNodeOfOtherAvailableNodesEventListener(InformNewNodeOfOtherAvailableNodesEventListener);
1832
1833 PackageVirtualNetworksManagerAgent_UpdateAvailableNodesListEventListener
UpdateAvailableNodesListEventListener = new
PackageVirtualNetworksManagerAgent_UpdateAvailableNodesListEventHandler();
1834 SystemMaster.AddUpdateAvailableNodesListEventListener(UpdateAvailableNodesListEventListener);
1835
1836 PackageVirtualNetworksManagerAgent_RequestingAllAvailableNodesEventListener
RequestingAllAvailableNodesEventListener = new
PackageVirtualNetworksManagerAgent_RequestingAllAvailableNodesEventHandler();
1837 SystemMaster.AddRequestingAllAvailableNodesEventListener(RequestingAllAvailableNodesEventListener);
1839  PackageVirtualNetworksManagerAgent_CreateNewVirtualNetworkOnEntireNetworkEventListener CreateNewVirtualNetworkOnEntireNetworkEventListener = new PackageVirtualNetworksManagerAgent_CreateNewVirtualNetworkOnEntireNetworkEventHandler();
1840  SystemMaster.AddCreateNewVirtualNetworkOnEntireNetworkEventListener(CreateNewVirtualNetworkOnEntireNetworkEventListener);
1841  
1842  PackageVirtualNetworksManagerAgent_CreateVirtualLinkEventListener CreateVirtualLinkEventListener = new PackageVirtualNetworksManagerAgent_CreateVirtualLinkEventHandler();
1843  SystemMaster.AddCreateVirtualLinkEventListener(CreateVirtualLinkEventListener);
1844  
1845  PackageVirtualNetworksManagerAgent_SetUpVirtualNetworkEnvironmentEventListener SetUpVirtualNetworkEnvironmentEventListener = new PackageVirtualNetworksManagerAgent_SetUpVirtualNetworkEnvironmentEventHandler();
1846  SystemMaster.AddSetUpVirtualNetworkEnvironmentEventListener(SetUpVirtualNetworkEnvironmentEventListener);
1847  
1848  PackageVirtualNetworksManagerAgent_DeleteVirtualLinkEventListener DeleteVirtualLinkEventListener = new PackageVirtualNetworksManagerAgent_DeleteVirtualLinkEventHandler();
1849  SystemMaster.AddDeleteVirtualLinkEventListener(DeleteVirtualLinkEventListener);
1850  
1851  PackageVirtualNetworksManagerAgent_SetUpVirtualNetworkUsingTextFileEventListener SetUpVirtualNetworkUsingTextFileEventListener = new PackageVirtualNetworksManagerAgent_SetUpVirtualNetworkUsingTextFileEventHandler();
1852  SystemMaster.AddSetUpVirtualNetworkUsingTextFileEventListener(SetUpVirtualNetworkUsingTextFileEventListener);
1853  
1854  PackageVirtualNetworksManagerAgent_DeleteVirtualNetworkOnEntireNetworkEventListener DeleteVirtualNetworkOnEntireNetworkEventListener = new PackageVirtualNetworksManagerAgent_DeleteVirtualNetworkOnEntireNetworkEventHandler();
1855  SystemMaster.AddDeleteVirtualNetworkOnEntireNetworkEventListener(DeleteVirtualNetworkOnEntireNetworkEventListener);
1856  
1857  
1858  
1859 ="/******************************************************/
1860  /*** 'holonic technology platform interface' thread ***/
1861  
1862  //initializing thread that will run 'holonic technology platform interface'
1863  Runnable RunHolonicTechnologyPlatformInterfaceThread = new Runnable()
1864  {
1865      public void run() //thread will be told which particular function to run
1866      {
1867          System.out.println("DEVELOPER INFORMATION: 'Holonic Technology Platform Interface' thread has been initiated.");
while(!(SystemMaster.StopThreads()))
{
    SystemMaster.HolonicTechnologyPlatformInterfaceApplication();
}

System.out.println("DEVELOPER INFORMATION: 'Holonic Technology Platform Interface' thread is now being terminated.");
System.exit(0);

lcreate a new thread and call it Holonic Technology Platform Interface Thread
Thread HolonicTechnologyPlatformInterfaceThread = new Thread(RunHolonicTechnologyPlatformInterfaceThread, "Holonic Technology Platform Interface Thread");

//start thread
HolonicTechnologyPlatformInterfaceThread.start();

/*** 'listen for application layer messages' thread **/

//initializing thread that will run 'listen for application layer messages'
Runnable RunListenForApplicationLayerMessagesThread = new Runnable()
{
    public void run() //thread will be told which particular function to run
    {
        System.out.println("DEVELOPER INFORMATION: 'Listen For Application Layer Messages' thread has been initiated.");
        while(!(SystemMaster.StopThreads()))
        {
            SystemMaster.ListenForApplicationLayerMessagesApplication();
        }
        System.exit(0);
    }
};

//create a new thread and call it Listen For Application Layer Messages Thread
Thread ListenForApplicationLayerMessagesThread = new Thread(RunListenForApplicationLayerMessagesThread, "Listen For Application Layer Messages Thread");

//start thread
ListenForApplicationLayerMessagesThread.start();
//start the thread

1911
1912
1913 //******************************************************************************
1914 /** listen for communications layer messages' thread
1915 /**
1916 //*****************************************************************************/
1917
1918 //initializing thread that will run 'listen for
1919 Communications layer messages'
1920
1921 Runnable RunListenForCommunicationsLayerMessagesThread =
1922 new Runnable()
1923 {
1924 public void run() //thread will be told which
1925 particular function to run
1926 {
1927 System.out.println("DEVELOPER
1928 INFORMATION: 'Listen For Communications Layer Messages' thread has been initiated.");
1929 while(!(SystemMaster.StopThreads()))
1930 {
1931 SystemMaster.ListenForCommunicationsLayer
1932 MessagesApplication();
1933 }
1934 System.out.println("DEVELOPER INFORMATION: 'Listen For
1935 Communications Layer Messages' thread is now
1936 being terminated.");
1937 System.exit(0);
1938 }
1939
1940 //create a new thread and call it Listen For
1941 Communications Layer Messages Thread
1942 Thread ListenForCommunicationsLayerMessagesThread = new
1943 Thread(RunListenForCommunicationsLayerMessagesThread, "Listen For
1944 Communications Layer Messages Thread");
1945
1946 //start thread
1947 ListenForCommunicationsLayerMessagesThread.start();
1948 //start the thread
1949
1950 //******************************************************************************
1951 <-- garbage testing -->
1952 //******************************************************************************
1953
1954 PackageAbstractComponents_AbstractMessage DummyMessage =
1955 new PackageAbstractComponents_AbstractMessage();
1956
1957 PackageAbstractComponents_AbstractInputOutputQueue
1958 TempMainInputQueue = SystemMaster.GetMainInputQueue();
1959
1960 TempMainInputQueue.ModifyQueue(1, DummyMessage,
"testing1");
1951 System.out.println("size: "+
1952 SystemMaster.GetMainInputQueue().size());
1953 TempMainInputQueue.ModifyQueue(1, DummyMessage, "testing2");
1954 System.out.println("size: "+
1955 SystemMaster.GetMainInputQueue().size());
1956 System.out.println("size: "+
1957 SystemMaster.GetMainInputQueue().size());
1958 int apple = 0;
1959 */
1960 }
1961 *
1962 /* software transfer functionality */
1963 ******************************************
1964 public static void TestIfSoftwareHasBeenTransferredSuccessfully()
1965 {
1966 System.out.println("SYSTEM INFORMATION: The following
class, HolonicTechnologyPlatformMaster, has been loaded successfully
after the transfer of byte codes.");
1967 }
1968 */
1969 /**********
1970 /* private classes */
1971 /**********
1972
1973 /**********
1974 /* private classes */
1975 /**********
1976
1977 private class TimerUpFunction extends TimerTask
1978 {
1979 public void run()
1980 {
1983 }
1984 }
1985
1986 private class BackupQueueForSendingSplitUpMessagesTimerUpFunction extends TimerTask
1987 {
1988 public void run()
1989 {
1992 }
1993 }
1994
1995 private class RequestingAvailableNodesTimerUpFunction extends TimerTask
1996
public void run()
{
    HolonicTechnologyPlatformMaster.FireSendAvailableNodesListToIntelligenceLayerEvent();
}

cpyivate class SynchronizeWithAtomicClockFunction extends TimerTask
{
    public void run()
    {
        String Hostname = "time.nist.gov";
        try
        {
            Socket HostSocket = new Socket(Hostname, 13);
            InputStream TimeInputStream = HostSocket.getInputStream();
            StringBuffer Time = new StringBuffer();
            int c;
            while ((c = TimeInputStream.read()) != -1)
            {
                Time.append((char) c);
            }
            String TimeString = Time.toString().trim();
            String UpdatedMessage = "SYSTEM INFORMATION: It is " + TimeString + " at " + Hostname;
            HolonicTechnologyPlatformMaster.FireOutputToConsoleEvent(UpdatedMessage);
            HolonicTechnologyPlatformMaster.SetSystemTimeStamp(TimeString);
            catch (UnknownHostException ex)
           {
                System.err.println(ex);
            }
            catch (IOException ex)
           {
                System.err.println(ex);
            }
        }
    }
}
private class AutosaveFunction extends TimerTask {
    public void run()
    {
        HolonicTechnologyPlatformMaster.FireInitiateSaveAndQuitEvent(0);
    }
}

# GLOSSARY

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent</td>
<td>A physical or virtual entity whose behaviour leads to satisfying its objectives.</td>
</tr>
<tr>
<td>Centralized Network</td>
<td>A network where the resources and processing capabilities are only available in a central area of the network. An example is a client-server architecture.</td>
</tr>
<tr>
<td>Centralized System</td>
<td>A system where the resources and processing capabilities are only available in a central area of the system.</td>
</tr>
<tr>
<td>Decentralized System</td>
<td>A system where every node on the system has the same copy of information, and where every node on the system has the same processing capability.</td>
</tr>
<tr>
<td>Distributed Intelligence System (DIS)</td>
<td>A distributed system that has the ability to adapt to a changing network environment via the use of intelligent algorithms.</td>
</tr>
<tr>
<td>Distributed System (DS)</td>
<td>A system where every node on the system has a unique set of information, and where every node on the system has a unique processing capability. Through this distribution of information and processing capability, the sum of all the information and the sum of all the processing capabilities provide a complete set for the network.</td>
</tr>
<tr>
<td>Holon</td>
<td>An agent that has the capability to work with other holons in trying to achieve a global network goal. Although holons also strive to complete local goals, a holon will sacrifice the progress of completing its local goals if it means the holon can aid in achieving the global network goal more quickly.</td>
</tr>
<tr>
<td>Holonic Agent</td>
<td>An agent that exists in the holon's infrastructure.</td>
</tr>
<tr>
<td>Holonic Communications System (HCS)</td>
<td>The system that facilitates communications in the holonic environment.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Holonic Environment</td>
<td>A network of holons created by the Holonic Technology Platform.</td>
</tr>
<tr>
<td>Holonic Logistics System (HLS)</td>
<td>A system that manages the holonic environment.</td>
</tr>
<tr>
<td>Holonic Strategic System (HSS)</td>
<td>A system that intelligently analyzes the holonic environment's present state and decides how to adapt to the changes in the state.</td>
</tr>
<tr>
<td>Holonic Technology Platform (HTP)</td>
<td>A distributed intelligence framework that is based on a hybrid model of multi-agent systems, utilizing holons and agents to perform and satisfy both local and global tasks and goals.</td>
</tr>
<tr>
<td>Multi-Agent System</td>
<td>A system composed of multiple agents that are capable of mutual interaction.</td>
</tr>
<tr>
<td>Unified Modelling Language (UML)</td>
<td>A general-purpose notational language for specifying and visualizing complex software, especially large, object-oriented projects. UML builds on previous notational methods such as Booch, OMT, and OOSE.</td>
</tr>
</tbody>
</table>
REFERENCE LIST


COLOPHON

Software programs that were used to develop the project and to compile this thesis document:

1. Software development was performed using Eclipse SDK (Version 3.2.0) and JCreator Pro (Version 3.00). The Java environment used was JDK1.5.0_07.

2. This document was written using Microsoft® Office Word 2003 and the figures in this document were created using Microsoft® Office Visio® Professional 2003. TextPad (Version 4.7.3) and irfanView (Version 3.80) were programs used to aid in improving the aesthetics of the code listings and figures.