CONCURRENCY MODELING EXTENSIONS TO
THE FUSION DEVELOPMENT METHODOLOGY

by

PETER W. WENZEL, B.S.

A THESIS

IN

COMPUTER SCIENCE

Submitted to the Graduate Faculty of Texas Tech University in
Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

Approved

May, 1997
CHAPTER

1. INTRODUCTION ................................................................. 1

2. CONCURRENCY MODELING IN OBJECT-ORIENTED SYSTEMS ....... 4

  2.1 Recent Work in Object-Oriented Concurrency Modeling .......... 4
      2.1.1 Plessman and Tassakos ......................................... 4
      2.1.2 Object Relationship Diagrams .................................. 7
      2.1.3 Parallel Objects (PO) ........................................... 9
      2.1.4 uC++ .................................................................. 10
      2.1.5 OPNets ................................................................ 12
      2.1.6 CODARTS ............................................................ 12

  2.2 Established Methodologies .............................................. 14
      2.2.1 Rumbaugh (1991) ................................................ 14
      2.2.2 Shlaer/Mellor (1988/1992) ..................................... 16
      2.2.3 Firesmith (1993) .................................................. 16
      2.2.4 Booch (1994) ...................................................... 18

3. FUSION DEVELOPMENT METHODOLOGY ............................... 19

  3.1 Methodology ................................................................. 20
      3.1.1 Analysis .............................................................. 22
          3.1.1.1 Object Model ................................................ 23
          3.1.1.2 Interface Model ............................................. 26
          3.1.1.3 Analysis Model Checking ................................. 28
      3.1.2 Design ................................................................. 29
          3.1.2.1 Object Interaction Graphs ................................. 29
3.1.2.2 Visibility Graphs ........................................... 33
3.1.2.3 Class Descriptions ....................................... 35
3.1.2.4 Inheritance Graphs ....................................... 36
3.1.2.5 Design Model Checking .................................... 36

3.2 Deficiencies ..................................................... 37

4. CONCURRENCY MODELING EXTENSIONS TO FUSION ............ 38

4.1 Requirements for Concurrency Modeling ......................... 38
  4.1.1 Threads of Control ........................................ 38
  4.1.2 Mutual Exclusion .......................................... 39
  4.1.3 Communication and Synchronization .......................... 40
  4.1.4 Concurrency Modeling Extensions Summary .................. 41

4.2 Fusion Extensions ............................................... 41
  4.2.1 Threads of Control Extensions .............................. 42
    4.2.1.1 Threads of Control Notation ......................... 42
    4.2.1.2 Threads of Control Identification .................. 43
  4.2.2 Communication and Synchronization Extensions .............. 44
    4.2.2.1 Communication and Synchronization Notation .......... 44
    4.2.2.2 Communication and Synchronization Identification ... 45
  4.2.3 Mutual Exclusion Extensions ............................... 46
    4.2.3.1 Mutual Exclusion Notation .......................... 46
    4.2.3.2 Mutual Exclusion Identification .................... 48
  4.2.4 Fusion Process Extensions .................................. 48
    4.2.4.1 Object Interaction Graphs Process Extensions ......... 49
    4.2.4.2 Class Descriptions Process Extensions ............... 50
  4.2.5 Inheritance of Concurrency .................................. 50

5. CELLULAR DIGITAL PACKET DATA .................................. 51

5.1 OSI Network Model .............................................. 51

5.2 CDPD Network Model ............................................ 53
  5.2.1 Interface to Existing Networks ............................ 54
  5.2.2 Mobility Management Overview .............................. 56
    5.2.2.1 Mobile Data Intermediate System (MD-IS) .......... 57
    5.2.2.2 Mobile Data Base Station (MDBS) .................... 58
    5.2.2.3 Mobile End Systems (M-ESs) ........................ 58

iii
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.3 Air Data Link</td>
<td>59</td>
</tr>
<tr>
<td>5.3 CDPD Mobility Management</td>
<td>62</td>
</tr>
<tr>
<td>5.3.1 Mobile Home Function (MHF)</td>
<td>63</td>
</tr>
<tr>
<td>5.3.2 Mobile Serving Function (MSF)</td>
<td>64</td>
</tr>
<tr>
<td>5.3.3 Redirection and Forwarding</td>
<td>64</td>
</tr>
<tr>
<td>5.3.4 Mobile Network Location Protocol (MNLP)</td>
<td>65</td>
</tr>
<tr>
<td>5.3.4.1 MNLP Parameters</td>
<td>66</td>
</tr>
<tr>
<td>5.3.4.2 Forwarding Information Base</td>
<td>66</td>
</tr>
<tr>
<td>5.3.4.3 Location Update Function</td>
<td>67</td>
</tr>
<tr>
<td>5.3.4.4 Forward NPDU Function</td>
<td>72</td>
</tr>
<tr>
<td>5.3.5 Mobile Network Registration Protocol (MNRP)</td>
<td>74</td>
</tr>
<tr>
<td>5.3.5.1 MNRP Parameters</td>
<td>75</td>
</tr>
<tr>
<td>5.3.5.2 Report Configuration Function</td>
<td>75</td>
</tr>
<tr>
<td>5.3.5.3 Record Configuration Function</td>
<td>76</td>
</tr>
<tr>
<td>5.3.5.4 Flush Old Configuration Function</td>
<td>77</td>
</tr>
<tr>
<td>5.3.5.5 Query Configuration Function</td>
<td>78</td>
</tr>
<tr>
<td>6. CASE STUDY</td>
<td>80</td>
</tr>
<tr>
<td>6.1 Analysis</td>
<td>80</td>
</tr>
<tr>
<td>6.1.1 Object Model</td>
<td>80</td>
</tr>
<tr>
<td>6.1.1.1 System Object Model</td>
<td>94</td>
</tr>
<tr>
<td>6.1.2 Interface Model</td>
<td>97</td>
</tr>
<tr>
<td>6.1.2.1 Life-Cycle Model</td>
<td>98</td>
</tr>
<tr>
<td>6.1.2.2 Operational Model</td>
<td>99</td>
</tr>
<tr>
<td>6.2 Design</td>
<td>109</td>
</tr>
<tr>
<td>6.2.1 Object Interaction Graphs</td>
<td>109</td>
</tr>
<tr>
<td>6.2.1.1 MSF Object Interaction Graphs</td>
<td>110</td>
</tr>
<tr>
<td>6.2.1.2 MHF Object Interaction Graphs</td>
<td>116</td>
</tr>
<tr>
<td>6.2.2 Visibility Graphs</td>
<td>122</td>
</tr>
<tr>
<td>6.2.3 Class Descriptions</td>
<td>124</td>
</tr>
<tr>
<td>6.2.4 Inheritance Graphs</td>
<td>138</td>
</tr>
<tr>
<td>7. CONCLUSION</td>
<td>139</td>
</tr>
<tr>
<td>7.1 Future Research</td>
<td>139</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>141</td>
</tr>
</tbody>
</table>
ABSTRACT

The “Fusion” software development methodology is a self-claimed second-generation full-coverage development method for object-oriented software covering the traditional analysis, design, and implementation phases as well as providing management tools for software development. Fusion’s deficiency is its lack of support for concurrency modeling which is essential in the problem domains of all real-time systems. With this one exception, Fusion is an excellent example of a fully integrated object-oriented development methodology, combining the best of several first-generation object-oriented analysis and design (OOAD) methods. The Fusion development methodology may be extended by integrating concurrency modeling into the method, making it more suitable for real-time problem domains.

The goals of this thesis are threefold: (1) identify the requirements for modeling concurrency in object-oriented systems, (2) propose extensions to the Fusion object-oriented method for modeling concurrency, and (3) demonstrate the proposed concurrency modeling extensions via a case study. The thesis identifies basic object-oriented concurrency modeling requirements by examining existing concurrency modeling techniques. These requirements are then used to form highly integrated concurrency modeling extensions to the Fusion object-oriented development methodology. Finally, the Fusion concurrency modeling extensions are demonstrated using the telecommunications real-time problem domain of cellular digital packet data (CDPD).
<table>
<thead>
<tr>
<th>Table Number</th>
<th>Table Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>uC++ Fundamental Abstractions</td>
<td>11</td>
</tr>
<tr>
<td>3.1</td>
<td>Life-Cycle Model Expression</td>
<td>27</td>
</tr>
<tr>
<td>3.2</td>
<td>Operation Model Template</td>
<td>28</td>
</tr>
<tr>
<td>6.1</td>
<td>Protocol Connection Data Dictionary</td>
<td>83</td>
</tr>
<tr>
<td>6.2</td>
<td>Cellular Entities Data Dictionary</td>
<td>85</td>
</tr>
<tr>
<td>6.3</td>
<td>Physical Entities Data Dictionary</td>
<td>87</td>
</tr>
<tr>
<td>6.4</td>
<td>MD-IS Data Dictionary</td>
<td>90</td>
</tr>
<tr>
<td>6.5</td>
<td>MD-IS Data Dictionary Continued</td>
<td>91</td>
</tr>
<tr>
<td>6.6</td>
<td>MNRP Message Generalization Data Dictionary</td>
<td>93</td>
</tr>
<tr>
<td>6.7</td>
<td>MNLP Message Generalization Data Dictionary</td>
<td>94</td>
</tr>
<tr>
<td>6.8</td>
<td>Agents Data Dictionary</td>
<td>96</td>
</tr>
<tr>
<td>6.9</td>
<td>System Interface Data Dictionary</td>
<td>97</td>
</tr>
<tr>
<td>6.10</td>
<td>esh Schema</td>
<td>101</td>
</tr>
<tr>
<td>6.11</td>
<td>rdc Schema</td>
<td>102</td>
</tr>
<tr>
<td>6.12</td>
<td>rdr Schema</td>
<td>103</td>
</tr>
<tr>
<td>6.13</td>
<td>rdf Schema</td>
<td>104</td>
</tr>
<tr>
<td>6.14</td>
<td>esb Schema</td>
<td>105</td>
</tr>
<tr>
<td>6.15</td>
<td>msfHoldingTimerExpiry Schema</td>
<td>106</td>
</tr>
<tr>
<td>6.16</td>
<td>mhfHoldingTimerExpiry Schema</td>
<td>107</td>
</tr>
<tr>
<td>6.17</td>
<td>rde Schema</td>
<td>107</td>
</tr>
<tr>
<td>6.18</td>
<td>mhfNpdu Schema</td>
<td>108</td>
</tr>
</tbody>
</table>
6.19 msfNpdu Schema

------------------------- 108
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Send Primitive</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>ReliableSend Primitive</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>ExceptionalSend Primitive</td>
<td>6</td>
</tr>
<tr>
<td>2.4</td>
<td>Ask/Reply Primitives</td>
<td>7</td>
</tr>
<tr>
<td>2.5</td>
<td>Behavior Model Notation</td>
<td>8</td>
</tr>
<tr>
<td>2.6</td>
<td>Stack Behavior Mode</td>
<td>9</td>
</tr>
<tr>
<td>2.7</td>
<td>Concurrency</td>
<td>15</td>
</tr>
<tr>
<td>2.8</td>
<td>Synchronization</td>
<td>15</td>
</tr>
<tr>
<td>2.9</td>
<td>Firesmith Concurrency Modeling</td>
<td>18</td>
</tr>
<tr>
<td>3.1</td>
<td>Evolution of Fusion</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>Fusion Methodology Summary</td>
<td>22</td>
</tr>
<tr>
<td>3.3</td>
<td>Object Model Class</td>
<td>23</td>
</tr>
<tr>
<td>3.4</td>
<td>Object Model Relationship</td>
<td>23</td>
</tr>
<tr>
<td>3.5</td>
<td>Object Model Aggregation</td>
<td>24</td>
</tr>
<tr>
<td>3.6</td>
<td>Object Model Generalization</td>
<td>24</td>
</tr>
<tr>
<td>3.7</td>
<td>Object Model Cardinality</td>
<td>25</td>
</tr>
<tr>
<td>3.8</td>
<td>System Object Model Boundary</td>
<td>25</td>
</tr>
<tr>
<td>3.9</td>
<td>Object Model Total Marker</td>
<td>25</td>
</tr>
<tr>
<td>3.10</td>
<td>Object</td>
<td>30</td>
</tr>
<tr>
<td>3.11</td>
<td>Collection of Objects</td>
<td>30</td>
</tr>
<tr>
<td>3.12</td>
<td>Message Passing to Object</td>
<td>30</td>
</tr>
</tbody>
</table>
3.13 Message Passing to Objects in Collection ................................. 31
3.14 Returned Value from Message ............................................. 31
3.15 Sequencing Information ....................................................... 32
3.16 Dynamic Object Creation .................................................... 32
3.17 Class (Client) ................................................................. 33
3.18 Server Objects ................................................................. 33
3.19 Visibility Reference Arrows .................................................. 34
3.20 Server Lifetime Bindings ...................................................... 34
3.21 Exclusivity References ....................................................... 34
3.22 Class Template ............................................................... 35
3.22 Inheritance ................................................................. 36
4.1 Thread of Control Extensions ............................................... 43
4.2 Communication Extensions .................................................. 45
4.3 Class Description Syntax Extensions ...................................... 47
4.4 Extended Class Description Template ..................................... 47
4.5 Mutual Exclusion Example ................................................... 48
5.1 An Internetwork of Administrative Domains ............................ 52
5.2 CDPD Network Interfaces .................................................... 53
5.3 The CDPD Network as an Extension of Existing Networks .......... 55
5.4 CDPD Network Connected to External Entities ....................... 56
5.5 CDPD Network Reference Model .......................................... 57
5.6 TEI Assignment ............................................................. 60
5.7 TEI Removal ................................................................. 61
6.34  Class Description: locationDirectoryClass ........................................ 132
6.35  Class Description: locationDirectoryEntryClass ................................. 133
6.36  Class Description: rdrClass ............................................................. 134
6.37  Class Description: rdcClass ............................................................. 135
6.38  Class Description: rdfClass ............................................................. 135
6.39  Class Description: rdeClass ............................................................. 136
6.40  Class Description: iscClass ............................................................. 137
6.41  Class Description: mdlpNpduClass .................................................. 137
6.42  Class Description: msfNpduClass .................................................... 138
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDPD</td>
<td>Cellular Digital Packet Data</td>
</tr>
<tr>
<td>CLNP</td>
<td>Connectionless Network Protocol</td>
</tr>
<tr>
<td>CODARTS</td>
<td>Concurrent DARTS</td>
</tr>
<tr>
<td>ES</td>
<td>End System</td>
</tr>
<tr>
<td>ESB</td>
<td>End System Bye</td>
</tr>
<tr>
<td>ESH</td>
<td>End System Hello</td>
</tr>
<tr>
<td>ESQ</td>
<td>End System Query</td>
</tr>
<tr>
<td>F-ES</td>
<td>Fixed End System</td>
</tr>
<tr>
<td>FS</td>
<td>Forwarding Service</td>
</tr>
<tr>
<td>LUS</td>
<td>Location Update Service</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IS</td>
<td>Intermediate System</td>
</tr>
<tr>
<td>ISC</td>
<td>MD-IS Hello Confirm</td>
</tr>
<tr>
<td>M-ES</td>
<td>Mobile End System</td>
</tr>
<tr>
<td>MDBS</td>
<td>Mobile Data Base Station</td>
</tr>
<tr>
<td>MD-IS</td>
<td>Mobile Data Intermediate System</td>
</tr>
<tr>
<td>MDLP</td>
<td>Mobile Data Link Protocol</td>
</tr>
<tr>
<td>MHF</td>
<td>Mobile Home Function</td>
</tr>
<tr>
<td>MNLP</td>
<td>Mobile Network Location Protocol</td>
</tr>
<tr>
<td>MNRP</td>
<td>Mobile Location Protocol</td>
</tr>
<tr>
<td>MSF</td>
<td>Mobile Serving Function</td>
</tr>
<tr>
<td>NEI</td>
<td>Network Entity Identifier</td>
</tr>
<tr>
<td>NPDU</td>
<td>Network Protocol Data Unit</td>
</tr>
<tr>
<td>OMT</td>
<td>Object Modelling Technique</td>
</tr>
<tr>
<td>OOAD</td>
<td>Object-Oriented Analysis and Design</td>
</tr>
<tr>
<td>RDC</td>
<td>Redirect Confirm</td>
</tr>
<tr>
<td>RDE</td>
<td>Redirect Expiry</td>
</tr>
<tr>
<td>RDF</td>
<td>Redirect Flush</td>
</tr>
<tr>
<td>Acronym</td>
<td>Full Form</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>RDQ</td>
<td>Redirect Query</td>
</tr>
<tr>
<td>RDR</td>
<td>Redirect Request</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
<tr>
<td>RSC</td>
<td>Registration Sequence Count</td>
</tr>
<tr>
<td>SNDCP</td>
<td>Subnetwork Dependent Convergence Protocol</td>
</tr>
<tr>
<td>SNPA</td>
<td>Subnetwork Point of Attachment</td>
</tr>
</tbody>
</table>
Successful concurrency modeling is essential in the problem domains of all real-time systems. This concept has long been accepted but until recently, concurrency issues have been relegated primarily to the implementation phase of the software life cycle. However, as systems become more and more complex, the implementation phase becomes too late to address concurrency issues for the first time. Concurrency modeling thus becomes part of the analysis and design phases of the software life cycle as well.

While structured methods were the first to address these concurrency issues, it is in the object-oriented paradigm’s intuitive concepts for modeling the real world that concurrency modeling has now moved. The object-oriented view is regarded as superior to others in its support for encapsulation, reusability, and extensibility. It may also provide a superior base on which to model concurrency; however, as Firesmith [35] notes, “… concurrency and real-time are often ignored or under emphasized in the object-oriented community which has tended to emphasize static structure and reuse over dynamic behavior design” (p. 27).

The goals for this thesis are threefold: (1) identify the requirements for modeling concurrency in object-oriented systems, (2) propose extensions to an existing object-oriented method for modeling concurrency, and (3) demonstrate the proposed concurrency modeling extensions via a case study. As a result of the author’s industry experience, this thesis is particularly interested in the analysis and design of real-time applications found in the telecommunications and data communications fields, as the case study covering the domain of Cellular Digital Packet Data will reveal.

The object-oriented design method selected for concurrency extensions is the Fusion development methodology of Coleman et al. [25]. Fusion is an excellent example of a fully integrated object-oriented development methodology, combining the best of several first-generation OOAD methods. The Fusion development methodology may be extended by integrating concurrency modeling into the method, making it suitable for the telecommunications and data communications problem domains. Wenzel [95] identifies a
shortcoming of most object-oriented methodologies in the description of system behavior which Fusion addresses quite adequately.

While the concepts behind object-orientation are becoming universally accepted, the terminology used to refer to these concepts has not yet reached a consensus. Before beginning to explore concurrency modeling amongst the various existing and recent methodologies, it is important to approach the object-oriented arena with a set of common terms. To this end, a brief overview of object-orientation, including definitions of the terms used throughout this thesis document, is provided in this introduction.

The basic building block of the object-oriented system is, of course, the object. In simplest terms, an object is anything that can be named. They can represent concepts, abstractions, or concrete things. Objects communicate with each other to perform complex tasks by sending messages to each other. These messages result in the invocation of the receiving object’s corresponding method, which then performs the requested action. The concept of encapsulation states that the sender of the message does not need to know how the object organizes its internal state, or attributes, only that the receiver has a well defined interface of messages that behave in a well defined way.

A class is considered a set of objects which share a common structure and behavior. An object is an instance of a class. In an object-oriented system, classes may have several different relationships between themselves. The term aggregation refers to a relationship in which a new class is constructed from several existing classes. The term inheritance refers to a relationship in which a class shares the structure and behavior of one (single inheritance) or more (multiple inheritance) other classes.

This document is divided into the following chapters. Chapter 2 covers the literature review, searching for existing object-oriented concurrency modeling techniques. Chapter 3 summarizes the Fusion development methodology as it currently exists. Chapter 4 draws on Chapter 2 in the identification of concurrency modeling requirements as well as Chapter 3 in making proposed extensions to Fusion for modeling concurrency in object-oriented systems. Chapter 5 serves as an introduction to the case study problem domain of Cellular Digital Packet Data. Chapter 6 contains the case study using Cellular Digital Packet Data as the vehicle for demonstrating Chapter 4’s proposed concurrency
modeling extensions to Fusion. Finally, Chapter 7 contains the conclusions drawn from the study and suggestions for future work.
CHAPTER 2
CONCURRENCY MODELING IN OBJECT-ORIENTED SYSTEMS

This chapter serves as a review of the current state of concurrency modeling in the object-oriented arena. Several established development methodologies such as Rumbaugh [86] and Booch [13] are surveyed as are newer developments in object-oriented concurrency from the detailed design and implementation phases of the software life-cycle. We first examine several new developments from this field. Following this we cover several existing software engineering methodologies which attempt to acknowledge concurrency in one form or another.

2.1 Recent Work in Object-Oriented Concurrency Modeling

Owing to the newness of object-oriented concurrency modeling, the vast majority of relevant work is originating from the programming language field. However, the concepts covered are also relevant to the analysis and design phases and several of the methods presented here do provide support for the detailed design phases as well. Presented in the following sections are the work of Plessman and Tassakos [81], Object Relationship Diagrams [52], Parallel Objects (PO) [27], uC++ [14], OPNets [56], and CODARTS [36].

2.1.1 Plessman and Tassakos

Plessman and Tassakos [81] offers an early view of concurrency in object-oriented programming. The authors stress the active character of real-time systems and proposes a model which contains parallel, event-driven activities of objects as its main emphasis. As per the standard object-oriented approach, each object in the model represents a separate unit which encapsulates information and the associated manipulation routines. The model departs from the typical Smalltalk-like object and moves toward the Actor model by optionally regarding an object as an active, autonomous actor in a system capable of operating concurrently either independently or cooperating with other objects. To avoid internal interferences within the objects themselves, each object is limited to performing
only one action at a given point of time. The concurrency in this model arises solely from the existence of multiple objects.

Inter-object communication occurs via messages using four basic primitives: send, reliableSend, ask, and reply. The send command allows fully asynchronous communication. After the sender has constructed the message and sent it, it can continue with other activities. Figure 2.1 shows a message flow diagram for the asynchronous send primitive.

Asynchronous messaging, while simple, does introduce problems with the guarantee of message delivery and the preservation of message ordering. The reliableSend primitive is introduced to solve these problems. A sender using reliableSend blocks all further activities until it receives an acknowledgment of the message being received. A derivative of the reliableSend known as the exceptionalSend provides for priority messaging by interrupting the receiver of the message. Figure 2.2 on page 6 shows a message flow diagram for the reliableSend primitive.
FIGURE 2.2 ReliableSend Primitive

Figure 2.3 shows a message flow diagram for the exceptionalSend primitive.

FIGURE 2.3 ExceptionalSend Primitive

Truly synchronous messaging is provided by the ask and reply primitives. Using the ask primitive, the sending object blocks not only until the receiving object has received the message but until the receiving object has processed the message to its
entirety and returned a result by means of the reply primitive. Figure 2.4 shows a message flow diagram for the ask/reply primitives.

2.1.2 Object Relationship Diagrams

Object Relationship Diagrams proposed by Kim and Moon [52] are based on Petri nets and are useful for describing the structural and behavior aspects of object-oriented systems at a conceptual level. The creators of Object Relationship diagrams emphasize the importance of modeling concurrency in object-oriented systems and define the two major characteristics of computing systems as "communication between objects" and "sharing of data between multiple concurrent activities" (p. 208).

Concurrency is divided into two distinct parts: inter-object and intra-object. In inter-object concurrency each object has a single activity executing one message at a time. The authors contend that traditional object-oriented modeling techniques are sufficient for modeling inter-object concurrency since message passing mechanisms are able to provide the necessary communication and synchronization between objects. However, traditional modeling techniques lack the features required to model intra-object concurrency.
Object Relationship Diagrams are presented as a means of modeling both inter- and intra-object concurrency. The diagrams consist of both structure diagrams and behavior diagrams to capture the structural and behavioral aspects of objects respectively. Structure diagrams describe the static aspects of objects using standard entity-relationship diagrams similar to [86]. Behavior diagrams build on the structural model to describe the synchronization required for both inter-object and intra-object concurrency.

Inter-object concurrency is modeling using what is called matching-events in which a set of event names define the events that can be executed in a certain state of an object. All other events are queued and the sender is blocked until such time as the object transitions to a new state and the queue is re-assessed for queued matching-events.

Intra-object concurrency is modeled based on mutual exclusion of shared data represented by a token in Petri nets. Figure 2.5 gives the notation for Object Relationship Diagram behavioral models and Figure 2.6 on page 9 shows a stack represented using Object Relationship Diagrams. Note that inter-object concurrency is represented with the matching-events while the Petri net token handles the intra-object concurrency.

![Object Relationship Diagram](image)

**FIGURE 2.5 Behavior Model Notation**
2.1.3 Parallel Objects (PO)

The Parallel Objects model [27] begins by recognizing two different dimensions of parallelism: inter-object and intra-object parallelism. Inter-object parallelism derives from the simultaneous presence of objects as “separate entities capable of execution and communications” (p. 403). Intra-object parallelism, on the other hand, provides “concurrency within the same object” (p. 404).

As a complete concurrent object system, communications in PO between objects can be either synchronous or asynchronous, however, the strength of PO lies in its intra-object concurrency modeling techniques. An object or method within an object in PO can be qualified by a series of scheduling constraints which must be satisfied before activating its operations. Constraints include maximally parallel (MaxPar), maximally sequential (MaxSeq), mutual exclusion (MutEx), priority (Pri), order (Ord), schedule condition (SchedCond), and post-action (PostAction).

The MaxPar constraint dictates that as soon as a request arrives, the scheduler spawns an activity dedicated to it. Objects handled by the MaxPar strategy do not limit the number of concurrent activities in anyway. MaxSeq objects require that only one request (the first in the queue) be handled at a time. MutEx scheduling applies to activities...
generated for the same operation. Only one activity associated with the same operation can be in execution in the object at any one time. The MutEx is particularly useful when applied to individual operations. MutEx(Op) allows only one activation of Op at a time within an object. MutEx(Op1,Op2) expresses mutual exclusion between methods Op1 and Op2. No activation of Op1 is possible as long as any activation of Op2 is in execution, and vice versa. As an example, a simple object with Read and Write methods would typically have the following scheduling constraints: MaxPar(Read), MaxSeq(Write), and MutEx(Read, Write).

The priority constraint Pri(Op1,Op2) gives higher priority to method Op1 than to Op2. Preemption of Op2 occur when a request for Op1 arrives. The order constraint Ord(Op1,Op2) forces an order between the activations of method Op1 and Op2. PO also allows guards on the scheduling of operations with the SchedCond(Op, <Condition>) condition and definition of post-actions with the PostAct(Op, <Action>) condition.

2.1.4 uC++

uC++, or “mu” C++, [14] is a dialect of C++ which introduces several new constructs to the language in support of concurrency. Although uC++ is an implementation language, the constructs introduced for concurrency shed light on concurrency modeling in general and are worth examination.

The creators of uC++ begin by defining three “elementary execution properties” on which their extensions are based. First is the concept of a thread. A thread is defined as the execution of code independently of other execution. According to the authors, a thread can be thought of as a virtual processor whose function is to advance execution by changing execution state. Multiple threads provide concurrent execution. uC++ contains constructs for the creation of new threads and for the control of how threads are used to accomplish computation. Additionally, constructs are provided which causes threads to block and subsequently be made ready for execution.

The second execution property is that of execution-state. An execution-state is the state information needed to permit concurrent execution, even if it is not used for
concurrent execution. An execution-state is either active or inactive, depending upon whether or not it is currently being used by a thread.

The final execution property is mutual exclusion. This is the mechanism that permits an action to be performed on a resource without interruption by other actions. In concurrent systems, mutual exclusion is required to guarantee consistent generation of results.

These three execution properties are then combined in various ways and applied to objects as properties of that object. Thread and execution-state refer to the object as a whole whereas mutual exclusion applies directly to the member functions of an object. Different combinations of the three properties produce five different kinds of objects. Table 2.1 summarizes the various combinations and the terms given them by uC++’s creators.

<table>
<thead>
<tr>
<th>Object Properties</th>
<th>Object’s Member Routine Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td>Execution State</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

The first abstraction identified by the matrix is that of the standard class-object. The class-object possesses none of the execution properties and is typically accessed only by a single task. The addition of mutual exclusion properties to the class-object’s member routines creates the second abstraction, that of the classic monitor originally defined in [42]. Objects with their own execution-state but no thread are known as coroutines and coroutine-monitors depending upon their member routines mutual exclusion properties. These objects use their callers thread to advance their own execution state. The final
abstraction is that of the task, encompassing its own thread, execution-state, and mutual exclusion properties.

uC++ supports various constructs for the creation and termination of coroutines and tasks, suspending and resuming their execution, and synchronization constructs for waiting on and blocking threads within mutex objects.

2.1.5 OPNets

The OPNet model proposed in [56] uses high-level Petri nets to represent the behavior of each object and the inter-object relationships. OPNets organizes a target system into concurrent objects and establish message passing relationships between those objects.

Two types of objects are defined in OPNets: primitive and composite. A primitive object is the basic entity for behavior representation in which static properties and dynamic behaviors are defined. Concurrency is not allowed in a primitive object since it is designed for representing sequential behaviors and static properties.

A composite object is an aggregate of more than one primitive object and/or composite objects. Concurrency can occur in a composite object because it structures primitive objects that have autonomous behavior. To synchronize the sequential behaviors of primitive objects, composite objects define message communications among its member objects.

2.1.6 CODARTS

The concurrent DARTS or CODARTS [36] system is an extension of the existing Design Approach for Real-Time Systems (DARTS). CODARTS focuses on concurrency, modularity, and finite state machines for developing large-scale concurrent real-time systems. This summary is concerned with the concurrency modeling of CODARTS.

Concurrency is achieved by having a design consisting of several concurrent tasks, each dealing with one sequential thread of execution. No concurrency is allowed within a task. The key feature of CODARTS concurrency are the principles provided for decomposing a real-time system into concurrent tasks and objects. The principles or
task-structuring criteria are grouped into four categories based on how they are used to assist in the task structuring: I/O, internal, cohesion, and priority. Note that a task may exhibit more than one of the criteria.

The first group of criteria examines how and when an I/O-dependent task is activated. Tasks included in this group are the Asynchronous I/O task, the Periodic I/O task, and the Resource Monitor task. For each asynchronous I/O device, there needs to be a separate asynchronous I/O device task to interface to it. A periodic I/O task is required for every passive I/O device. This task is activated periodically to either read or write to a passive device. Finally, any I/O device that receives requests from multiple sources needs a resource monitor task to coordinate these requests.

The next group of criteria known as the internal task-structuring criteria address how and when an internal task is activated. Tasks included in this group are the Periodic task, the Asynchronous task, the Control task, and the User Role task. Periodic tasks include any function that needs to be executed at regular intervals of time. Asynchronous tasks include functions that execute on demand by an event or message sent by another task. Control tasks include any control activities whose activity is defined by a state transition diagram. User role tasks include any sets of sequential operations performed by the user.

Task cohesion criteria provide a means for determining whether objects and/or functions should be combined into concurrent tasks. Cohesion criteria included in this group are: sequential, temporal, functional, control, and task inversion. Sequential cohesion describes functions that must be carried out in sequence. The first function in the sequence is triggered by an asynchronous or periodic event. The sequentially dependent functions may be combined into one sequentially cohesive task. Temporal cohesion describe functions that are activated at the same time. Functional cohesion describes functions grouped into a task because they perform a set of closely related operations that, due to application constraints, can not be executed simultaneously. Control cohesion occurs when a control object is executing a state transition diagram. Any functions required which can not execute concurrently with the control object may be combined into the control object based on control cohesion. Finally, task inversion occurs when
several tasks are combined simply because the system overhead of multiple tasks is too high.

Task priority criteria address the time criticality of tasks. Time critical functions should be structured into separate high-priority tasks. Non-time critical functions may be run as low priority tasks consuming spare CPU cycles.

2.2 Established Methodologies

The methodologies covered here are all considered complete object-oriented development schemes of varying tenure and acceptance by the object-oriented community. The methods are presented in descending order of age with the newest covered last as to show a possible evolution in object-oriented concurrency modeling at the analysis and early design stages.

Note that several existing methodologies pay little or no attention to concurrency modeling and thus have little add to the discussions presented here. These methods include Coad [20, 21] and Wirfs-Brock [99].

2.2.1 Rumbaugh (1991)

The Object Modeling Technique (OMT) of Rumbaugh [86] addresses concurrency first in the analysis phase while constructing the dynamic model. The dynamic model is based on state diagrams and each object is considered concurrent with its own state transition diagram. Concurrency within an object is also modeled via state diagrams.
capable of modeling parallelism and synchronization. Figure 2.7 shows the notation used for concurrency and Figure 2.8 shows synchronization.

OMT also marks the identification of concurrency as the second step in the system design phase. Note that in the analysis phase, all objects are considered concurrent. The goals of this design step are to identify which objects must be active concurrently and which objects have activity that is mutually exclusive with the mutually exclusive objects being combined into a single thread of control or task. Unfortunately, these guidelines are not directly supported by any particular notations or details and function only as heuristics for design of object-oriented systems [37].
2.2.2 Shlaer/Mellor (1988/1992)

Although the validity of the Shlaer/Mellor method presented in [87] and [89] as a true object-oriented method is under question [37], it does mention the issue of concurrency and provide for two different interpretations for dealing with it. The simultaneous interpretation allows two or more actions to be happening at the same time. The interleaved interpretation allows for only one thing to be happening at a time. We are concerned with how the method addresses the simultaneous interpretation only. Several rules are developed to aid with dealing with concurrency:

1. Only one method of a given object can be in execution at any point in time.
2. Methods in different objects can be executing simultaneously.
3. One initiated, a method of an object must complete before another message can be accepted by the same object.
4. When an object completes a method the object takes an available message directed at it if any such message exists at that time.
5. Multiple messages may be outstanding for a given object.

The method also allows for asynchronous messaging depicted in the Object Communication Model (OCM) and synchronous communication depicted in the Object Access Model. Characteristic of early object-oriented development methodologies, the method has very little support for concurrency as is acknowledged by its creators by the statement in [89]: "... concurrency and instance sharing are receiving significant attention as research topics. We expect it to be some time before widely applicable techniques for dealing with these issues become well-understood" (p. 195).

2.2.3 Firesmith (1993)

Firesmith [13] approaches concurrency early in the analysis and logical design (the precursor to implementation design) phases of the development life-cycle. Concurrency is addressed in terms of both objects, which may be concurrent or sequential, and messages, which may be sequential, synchronous, or asynchronous.

Concurrent objects have their own thread of control or contain resources such as operations or sub-objects that have their own thread of control. Firesmith argues that most
objects during the analysis and logical design phases should be concurrent as most of the real world entities they model are concurrent. Sequential objects do not have their own thread of control. Mutual exclusion is identified as a critical issue when dealing with multiple threads of control. Objects are categorized as either corruptible, in which the object does not address mutual exclusion and is typically sequential in nature, guardable, in which the object permits mutually exclusive use (e.g., via binary semaphores) but does not enforce or guarantee mutual exclusion, and guarded, in which an object enforces and guarantees mutually exclusive access to its resources (e.g., via monitors).

In addition to the objects themselves, the operations within the objects are also categorized with regards to concurrency. Sequential operations do not have their own thread of control and exist only in sequential objects. Concurrent operations on the other hand have their own thread of control.

Communications between objects are handled strictly via message passing. Messages may be either sequential, which involves only one thread of control and can only be sent to sequential objects; synchronous, which involves two threads of control which must synchronize during the execution of an operation and can only be sent to concurrent objects; and asynchronous, which involve no synchronization and can also only be sent to concurrent objects. The method also specifies flow of control for messages with special qualifiers being added to synchronous messages. Synchronous messages can be either standard, in which the sending object suspends until a response is received, balking, in which the sender immediately abandons the operation if the receiver is not immediately ready, and time-out, in which the sender waits only a specified period of time for the receiver to be ready before abandoning the operation.
Figure 2.9 outlines the notation used for representing some of these concepts. Note that the notation contains over a dozen different diagrams and several of the notations noted below are taken out of context of their original diagram.

2.2.4 Booch (1994)

Booch [13] defines concurrency as "... the property that distinguishes an active object from one that is not" (p. 513). Active objects are objects that embody their own thread of control. Conversely, passive objects do not have their own thread of control. Other types of objects include sequential, guarded, and synchronous. Sequential defines passive objects whose behavior is guaranteed only in the presence of one thread of control. A guarded object’s behavior is guaranteed in the presence of multiple threads of control with the requirement that all the threads cooperate to achieve mutual exclusion. Synchronous defines objects which guarantee their own mutual exclusion. The method also qualifies operations as either sequential, guarded, or synchronous.

Object communication and synchronization is performed via various types of messages. Messages are defined as either sequential, synchronous, balking, time-out, or asynchronous.
The Fusion development methodology, or simply, "Fusion," is a self-claimed second-generation full-coverage development method for object-oriented software covering the traditional analysis, design, and implementation phases as well as providing management tools for software development [25]. Two items are of particular interest in this statement which differentiate Fusion from the methodologies mentioned in Chapter 2. The first item of interest is the assertion that Fusion is a second-generation methodology. By this, we may assume that Fusion has borrowed from previous first-generation methodologies by integrating and extending the best of existing approaches, as is the claim of its creators. Figure 3.1 shows some of the existing methodologies to which Fusion owes its beginnings.

![FIGURE 3.1 Evolution of Fusion](image)

The second assertion of interest is that Fusion is a full-coverage methodology. This indicates that Fusion provides a direct and guided path from requirements definitions straight through to a language implementation. Again, it is the claim of its creators that
Fusion provides a complete process for software development. Fusion “divides the process into phases and indicates what should be done in each phase. It gives guidance on the order in which things should be done within phases, so the developer knows how to make progress. It provides criteria that tell the developer when to move on to the next phase” (p. 8). In addition, Fusion “provides management tools for software development,” “... the outputs different phases are clearly identified,” and “... there are cross-checks to ensure consistency within and between phases” (p. 9).

These characteristics give insight as to why Fusion has been adopted by the software houses of companies such as Hewlett-Packard and Northern Telecom (Nortel). It is not this author’s intention to demonstrate the advantages of Fusion over other object-oriented methodologies but rather to accept them at face value and proceed to build upon the methodology. This overview is considerably more detailed than those offered in Chapter 2, focusing on the entire analysis and design phases of the methodology rather than just the concurrency models provided. This detail is intended to help facilitate the explanation of the proposed concurrency extensions found in Chapter 4 and the demonstrative case study found in Chapter 6.

3.1 Methodology

As stated previously, Fusion’s methodology is broken into three phases, that of analysis, design, and implementation. Analysis is defined as that which “… produces a specification of what a system does … the intention is to provide a clear understanding of what the system is about and what its underlying concepts are … the result of the analysis phase is a specification document” (p. 4). Likewise, design, “… takes the specification document as its base and defines how the specified behavior is obtained from implementation-oriented components … the output of the design phase is an architecture document” (p. 4). Finally, implementation “… takes the design document and encodes the design in programming language … the output of the implementation phase is code” (p. 4).

It should be noted that Fusion’s “implementation” phase contains several steps that the software engineering community typically refers to as “detailed” or “low-level” design and is not relegated strictly to the production of code. A distinction should be made
between the Fusion “design” phase which more accurately equates to a “logical,”
“architectural,” or “high-level” design phase and the Fusion “implementation” phase
which does include some additional design considerations. Note that this overview, as
well as the concurrency extension of Chapter 4 and case study of Chapter 6, cover only
the Fusion analysis and design phases.

Fusion uses several models throughout the various phases to capture information.
Those used at analysis time are intended to capture the intended behavior of the system
and include models which describe classes of objects that exist in the system,
relationships between those classes, operations that can be performed on the system, and
allowable sequences of those operations. The design phase produces models which
capture how the system operations are implemented by interacting objects, how classes
refer to one another, how classes are related by inheritance, and the attributes of, and
operations on, classes. Figure 3.2 on page 22 shows a summary diagram of the entire
Fusion methodology.
3.1.1 Analysis

The analysis phase of Fusion produces two models which focus primarily on the domain of the problem and its externally visible behavior. The object model defines the static structure of the information in the problem domain. It is very similar to the object model of Rumbaugh’s OMT [86] with a few minor notational differences. The analysis phase then produces the interface model which describes the behavior of the system in terms of the input and output communications of the system.
3.1.1.1 Object Model

The purpose of the object model is to capture the concepts that exist in the domain of the problem and the relationships between them. It can represent classes, attributes, and relationships between classes. The object model notation is based on entity relationship diagrams with extensions for modeling aggregation and generalization in addition to standard classes, attributes, and relationships between classes. In summary, the object model consists of class boxes and relationship diamonds connected by subtypes and role arcs.

Figure 3.3 on page 23 through Figure 3.9 on page 25 give the object model notation in its entirety. See Chapter 6 for examples of this notation in use.

![FIGURE 3.3 Object Model Class](image1)

![FIGURE 3.4 Object Model Relationship](image2)
FIGURE 3.5 Object Model Aggregation

FIGURE 3.6 Object Model Generalization
FIGURE 3.7 Object Model Cardinality

FIGURE 3.8 System Object Model Boundary

FIGURE 3.9 Object Model Total Marker
The process for the creation of the object model is summarized as follows:

1. Brainstorm a list of candidate classes and relationships. Fusion does not offer a formal methodology for object or relationship identification.
2. Enter classes and relationships into the data dictionary.
3. Incrementally produce object model looking for:
   — Generalizations modeling “kind of” or is-a relationships.
   — Aggregations modeling “part of” or has-a relationships.
   — Attributes of classes.
   — Cardinalities of relationships.
   — General constraints that should be recorded in the data dictionary.
   — Derived relationships that should be recorded in the data dictionary but that do not appear on the object model.

The object model is then analyzed to determine the system interface of the object model which allows the system object model to be created. The process for determining the interface boundary is summarized as follows:

1. Identify agents, system operations, and events.
2. Produce the system object model.
   — Using information from the system interface, identify classes and relationships on the object model that pertain to the state of the system.
   — Document system boundary to produce the system object model.

Useful techniques for establishing the interface boundary include focusing on scenarios of usage. Particular attention should be given to the agents who are involved and what they want the system to do.

3.1.1.2 Interface Model

The next step in the analysis phase is to develop the interface model which is comprised of the both the life-cycle model and an operational model. The life-cycle model
defines the allowable sequences of interactions in which a system may participate and is defined in terms of regular expressions, see Table 3.1.

TABLE 3.1 Life-Cycle Model Expression

<table>
<thead>
<tr>
<th>Concatenation</th>
<th>x.y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alternation</td>
<td>xly</td>
</tr>
<tr>
<td>Zero or more</td>
<td>x*</td>
</tr>
<tr>
<td>One or more</td>
<td>x+</td>
</tr>
<tr>
<td>Optional</td>
<td>[x]</td>
</tr>
<tr>
<td>Interleaving</td>
<td></td>
</tr>
<tr>
<td>Grouping</td>
<td>(x)</td>
</tr>
</tbody>
</table>

The process for the creation of the life-cycle model is summarized as follows:
1. Generalize scenarios and form named life-cycle expressions.
2. Combine life-cycle expressions to form the life-cycle model.

The operational model defines the semantics of each system operation in the system interface. For each system operation:
1. Develop the Assumes and Results clauses.
   - Describe each aspect of the result as a separate subclause of Results.
   - Use the life-cycle model to find the events that have to be output in Results.
   - Check that Results does not allow unwanted values.
   - Add relevant system object model invariants to Assumes and Results.
   - Ensure Assumes and Results are satisfiable.
   - Update data dictionary entries for system operations and events.
2. Extract Sends, Reads, and Changes clauses from the Results and Assumes.
Table 3.2 shows the template for the operational model.

### TABLE 3.2 Operation Model Template

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Reads</th>
<th>Changes</th>
<th>Sends</th>
<th>Assumes</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>operation identifier</td>
<td>&lt;text&gt; Description of operation</td>
<td>&lt;supplied values&gt; &lt;state components&gt;</td>
<td>&lt;supplied values&gt; &lt;state components&gt;</td>
<td>&lt;agent communications&gt;</td>
<td>&lt;assertions&gt; (preconditions)</td>
</tr>
</tbody>
</table>

#### 3.1.1.3 Analysis Model Checking

The final step of the analysis phase is to verify the consistency and completeness of the analysis models. Models are consistent when they do not contradict each other. Models are complete when they capture all the meaningful abstractions in the domain. The following guidelines are provided for checking the analysis:

1. Completeness against the requirements. The requirements document(s) should be re-read carefully. Verify that:
   - All possible scenarios are covered by the life-cycle.
   - All possible operations are covered by the schema.
   - All static information is captured by the system object model.
   - Any other information such as technical definitions and invariant constraints are in the data dictionary.

2. Simple Consistency. These checks deal with the areas of overlap between the models of analysis. Verify that:
   - All classes, relationships, and attributes mentioned in the object model appear in the system object model. All other concepts must be defined in the data dictionary.
   - The boundary of the system object model is consistent with the interface model.
— All the system operations in the life-cycle model have a schema.
— All identifiers in all models have entries in the data dictionary.

3. Semantic Consistency. These checks attempt to ensure that implications of the models are consistent.
— Output of events in the life-cycle model and operation model must be consistent. The schema for a system operation must generate the output events that follow it in the life-cycle model.
— The operation model must preserve the system object model invariant constraints. If there is an invariant concerning a relationship or class, then any operation that can change them must respect the invariant in its schema.
— Desk check scenarios using the schemas. Choose examples of scenarios and define the state changes that each should cause. Then “execute” the scenarios, using the schemas to define the behavior of each system operation. Check that the result is what is expected.

3.1.2 Design

There are four design models developed during Fusion’s design phase. The object interaction graph describes how objects interact at run-time to support the functionality specified in the operation model. The visibility graph describes the object communication paths. The class descriptions provide a specification of class interface, data attributes, object reference attributes, and method signatures for all classes in the system. Finally, inheritance graphs describe any class/subclass inheritance structures.

3.1.2.1 Object Interaction Graphs

The first step in Fusion’s design phase is to construct object interaction graphs showing how functionality is distributed across the objects of a system. Figure 3.10 on
Page 30 through Figure 3.16 on page 32 give the object interaction graph notation in its entirety. See Chapter 6 for examples of this notation.

**FIGURE 3.10 Object**

**FIGURE 3.11 Collection of Objects**

**FIGURE 3.12 Message Passing to Object**
FIGURE 3.13 Message Passing to Objects in Collection

FIGURE 3.14 Returned Value from Message
The process for the creation of the object interaction graph is summarized as follows:

1. Identify relevant objects involved in the computation.
2. Establish the role of each object participating in the computation.
   - Identify controller.
   - Identify collaborators.
3. Decide on messages between objects.
4. Record how the identified objects interact on an object interaction graph.
Verify the following:

1. Consistency with analysis models. Check that each of the classes in the system object model are represented in at least one object interaction graph.
2. Verification of functional effect. Check that the functional effect of each object interaction graph satisfies the specification of its system operation given in the operation model.

3.1.2.2 Visibility Graphs

For each class, a visibility graph using the notation shown in Figure 3.17 on page 33 through Figure 3.21 on page 34 is now constructed showing how the object-oriented system is structured to enable object communication.

![Figure 3.17 Class (Client)](image)

![Figure 3.18 Server Objects](image)
The process for the construction of visibility graphs is summarized as follows:

1. All the object interaction graphs are inspected. Each message on an object interaction graph implies that a visibility reference is needed from the client class to the server class.

2. Decide on the kind of visibility reference required taking into account:
   — Lifetime of reference.
   — Visibility of target object.
— Lifetime of target object.
— Mutability of target object.

3. Draw a visibility graph for each design object class verifying that for each relation on the system object model there exists a path of visibility for the corresponding classes on the visibility graphs and that no exclusive target objects or shared targets are referenced by more than one class.

3.1.2.3 Class Descriptions

Figure 3.22 shows the syntax for a Fusion class description.

```plaintext
class <ClassName> [isa <SuperClassNames>]
  // for each attribute
  [attribute] [Mutability]<a_name> :[Sharing][Binding]<Type>
  // for each method
  [method] <m_name> <arglist>[::<Type>]
endclass
```

FIGURE 3.22 Class Template

Class descriptions are the specifications from which coding begins. The descriptions specify the internal state and external interface of each class. Class description derive their information from the system object model, object interaction graphs, and visibility graphs and record the following:

1. Methods and parameters from the object interaction graphs.
2. Data attributes from the system object model and data dictionary.
3. Object attributes from the visibility graph for the class.
4. Inheritance information from the inheritance graphs, see Section 3.1.2.4 on page 36.

Verify the following:

1. All methods from the object interaction graphs are recorded.
2. All data attributes from the system object model are recorded.
3. All visibility references are recorded.
4. All inherited superclasses are recorded.

3.1.2.4 Inheritance Graphs

Figure 3.22 shows the inheritance graph notation used by Fusion. Inheritance graphs are constructed by examining the classes in search of commonalities and abstractions such as:

- Generalizations and specializations in the object model.
- Common methods in the object interaction graphs and class descriptions.
- Common visibility in the visibility graphs.

![Inheritance Graph Diagram]

**FIGURE 3.22 Inheritance Graph**

3.1.2.5 Design Model Checking

The final step of Fusion’s design phase is to update the class descriptions with any new inheritance information and perform the following checks:

1. System Object Model. Verify that the subtype relations are preserved.
2. Object Interaction Graphs. Verify that all classes are represented in an inheritance graph.
3. Visibility Graphs. Verify that all classes are represented in an inheritance graph. Abstract base classes can be defined for common structure between classes in the visibility graphs.

4. Class Descriptions. Verify that updated class descriptions implement all the functionality of the preliminary ones and respect the inheritance graphs.

3.2 Deficiencies

Fusion's one glaring deficiency is its lack of support for concurrency modeling. Coleman [25] suggests that Fusion can be used for applications which "... exhibit a very limited form of concurrency in which (1) concurrency is provided by the operating system, (2) processes are sequential programs, (3) there are a relatively small number of processes, and (4) there is very limited use of dynamic process creation" (p. 46). Even with these restrictions, Coleman claims that "the task of designing concurrent system architectures lies outside of the scope of this book" (p. 46). Chapter 4 attempts to alleviate some of these deficiencies.
Chapter 2 serves as a starting point for the proposed extensions to Fusion. It should be noted that the list of methodologies reviewed in Chapter 2 is not exhaustive. Several methods including Martin/Odell [67], Jacobson [46], de Champeaux [30], and Embly [33] are not covered simply in the interest of space. However, this is not of particular concern as the main emphasis of Chapter 2 is not to detail and elaborate upon every known method of object-oriented concurrency modeling but rather to highlight the concurrency issues generally addressed by the methods. From this survey, a list of concurrency issues is constructed and used as the set of minimum requirements for the proposed extensions to Fusion.

4.1 Requirements for Concurrency Modeling

One of the first conclusions to be drawn when reviewing current object-oriented concurrency models is that the concurrency issues involved are not all that different from the concurrency issues of non-object-oriented models. Hoare [42] and Ben-Ari [8] describe some of the same mutual exclusion and synchronization problems that exist in today’s object-oriented systems. The terminology may be slightly different, but the issues are basically the same. This proves helpful in the creation of a set of requirements for concurrency models by adding an additional 30 years of theory and practice to the recent object-oriented work from which to draw on when modeling object-oriented concurrency at the analysis and design phases. With that as an introduction, we will now move on to the three major requirements of concurrency, that of (1) threads of control, (2) mutual exclusion, and (3) communication/synchronization.

4.1.1 Threads of Control

The literature provides several complimentary definitions for “thread of control.” Booch [13] defines the concept as follows:
A single process. The start of a thread of control is the root from which independent dynamic action within a system occurs; a given system may have many simultaneous threads of control, some of which may dynamically come into existence and then cease to exist. Systems executing across multiple CPUs allow for truly concurrent threads of control, whereas systems running on a single CPU can only achieve the illusion of concurrent threads of control. (p. 519)

Rumbaugh [86] defines threads of control as follows:

A thread of control is a path through a set of state diagrams on which only a single object at a time is active. A thread remains within a state diagram until an object sends an event to another object and waits for another event. The thread passes to the receiver of an event until it eventually returns to the original object. The thread splits if the object sends an event and continues executing. (p. 202)

Firesmith [35] defines threads of control as “a theoretically possible sequence of operation calls and/or exception flows within objects or classes or among objects, classes, and other entities” (p. 489).

The literature refers to objects which encompass their own thread of control as either “Active” or “concurrent.” We will use the term “Active” to identify objects with their own thread of control and “Passive” to denote objects which do not have their own thread of control.

The ability to identify and denote an object as either active or passive is the first requirement of the Fusion concurrency modeling extensions.

4.1.2 Mutual Exclusion

Mutual exclusion is the property which ensures that an action performed on a shared resource can be completed without interruption. Mutual exclusion for such operations is essential for consistent operation of systems involving concurrency. Ben-Ari [8] asserts that the property of mutual exclusion is preserved when “two processes may not interleave certain sequences of instructions” (p. 21).

When considering mutual exclusion for shared resources in object-oriented systems, collectively referred to as intra-object concurrency in the literature, there exists several approaches for ensuring consistent operation. First, one can consider the mutual exclusion property for the entire object. The first and most simple approach is to allow an
unlimited number of threads to be present in an object at any time. These objects do not limit the number of concurrent activities in the system in any way and also do not guarantee mutually exclusive access to resources and thus are known as both maximally concurrent and corruptible in the literature.

The second approach is to restrict an object to executing only one operation at a time, effectively serializing the object. This is analogous to the monitor concept introduced by Hoare [42], limiting the number of threads present in the object at any one time to one, effectively ensuring mutually exclusive access to any resources. These objects are referred to as maximally sequential and guarded in the literature.

A variation on the maximally sequential object allows only one thread per operation present in an object at any one time, effectively limiting the number of threads to equal the number of operations in the object. These objects are referred to as mutually exclusive objects.

The concept of mutual exclusion can be applied to an entire object but may also be applied to the individual operations (i.e., methods) within an object. For example, considering the classic problem of a object containing a resource with operations for both reading from and writing to the resource. Applying the same concepts of maximally concurrent, maximally sequential, and mutually exclusive allows the read operation to be defined as maximally concurrent, the write operation to be maximally sequential, and the two operations to be mutually exclusive of each other.

The ability to identify and denote mutual exclusion properties for object operations is the second requirement of the Fusion concurrency modeling extensions.

4.1.3 Communication and Synchronization

The concept of mutual exclusion and the possibility that a server object may not be able to immediately service the request of a client object due to mutual exclusivity restrictions introduces the idea that objects must synchronize their activities. This synchronization is carried out by defining the behavior of the operations from the viewpoint of the messaging taking place between objects. The literature defines three basic types of synchronization behaviors: sequential, synchronous, and asynchronous.
Sequential messaging takes place in the presence of a single thread of control in which the thread is passed from the client object to a passive server object until the corresponding operation is completed.

Synchronous messaging may take place between two threads of control and thus two active objects. Synchronous messaging may also be necessary between active and passive objects. In this case, the passive object takes the thread of control from the active object. Synchronous messaging may be necessary in this case due to mutual exclusion properties which can be present in passive objects. Synchronous messaging may be truly synchronous, in which the two objects are involved in a rendezvous; balking, in which the client will rendezvous with the server object only if the server object is ready and waiting; and time-out, in which the client object may rendezvous with a server object, but will only wait for a specified amount of time.

The final type of messaging is asynchronous and defines an absence of synchronization. With asynchronous messaging, the two objects act entirely independently of each other and the client object may initiate the operation regardless of whether the server is expecting the message.

The ability to identify and denote synchronization between objects is the final requirement for the Fusion concurrency modeling extensions.

4.1.4 Concurrency Modeling Extensions Summary

To summarize, the requirements for the proposed concurrency modeling extensions to Fusion are as follows:

1. The ability to identify and denote an object as either active or passive.
2. The ability to identify and denote mutual exclusion properties for individual object operations.
3. The ability to identify and denote synchronization between objects.

4.2 Fusion Extensions

With these requirements as a basis, we now move on to the actual extensions to Fusion which will be used in the case study in Chapter 6. One final requirement for the
extensions is that they be integrated into the existing Fusion model as tightly and as
seamlessly as is possible. This includes the concept that Fusion guides the user through
the process, facilitating his progress at every step. For example, it is not enough to simply
add notations denoting active objects without providing rules and heuristics for
determining which objects are active. Additional models (i.e., new and separate diagrams)
are to be avoided when at all possible, with preference given to extensions to existing
models. In the following sections, several extensions to Fusion are presented for the
modeling of concurrency. The extensions focus first on notation and are then integrated
into the Fusion process.

4.2.1 Threads of Control Extensions

All Fusion classes in the analysis phase are considered active. The design phase,
however, does not currently allow the specification of objects which contain their own
thread of control. Extensions are needed to allow both active and passive objects.

Any object which possesses its own thread of control is considered active and
denoted as such in the design models with the keyword ‘Active’ preceding the object or
class name in the object interactions graphs and class descriptions. Passive objects, those
without a thread of control, are denoted as per normal Fusion notation (i.e., absence of the
‘Active’ keyword).

4.2.1.1 Threads of Control Notation

Figure 4.1 on page 43 shows an object interaction graph in which objects are
marked as either explicitly active or implicitly passive. The class descriptions are also
extended to show the active and passive nature of objects. These notational extensions are
included in Section 4.2.3 on page 46 which deals in more depth with modifications to the
class description syntax.
4.2.1.2 Threads of Control Identification

The identification of active objects is performed in the Fusion design phase during the construction of the object interaction graphs for the system. All objects which receive system operations must be active. Typically these objects are required to receive asynchronous messages from the outside environment. Also included in this category would be objects which perform periodic i/o with the outside world. Sensors in a process control domain qualify for this designation.

The designation of all objects which interact with the environment as active serves as the basis for the discovery of the remainder of the active objects in the system. Any active object which communicates with another object via internal asynchronous messaging, see Section 4.2.2 on page 44, requires the receiver of that message to also be an active object. Using this rule, the active objects of a system may be determined by following the asynchronous messaging throughout the system.

In addition, the synchronous messaging discovered in Section 4.2.2 on page 44, should be examined to determine additional active objects. Receivers of synchronous messages are not required to contain their own thread of control, although, obviously, they must eventually be implemented as separate processes. Synchronous messaging serves only as a guide and not a absolute method for identification of active objects.
4.2.2 Communication and Synchronization Extensions

Communication between objects in Fusion is first described in the design phase with the object interaction graphs. Traditional Fusion allows only sequential messaging, represented via arrows between objects, to take place. All messaging is considered to take place via method calls which must complete before control is returned to the caller. In the event that the invoked method also sends messages (i.e., method invocations) to other objects, then the methods that are invoked must all complete before the initial invocation completes. Extensions are needed to allow both synchronous and asynchronous communication as well as sequential communication.

4.2.2.1 Communication and Synchronization Notation

The following notations are introduced to denote synchronous and asynchronous messaging. Figure 4.2 on page 45 shows the use of various arrows to denote asynchronous, synchronous, and sequential communication. In this example, active object 1 receives an asynchronous event from the environment which results in a time-out synchronous message (1) being sent to active object 2 at which point it waits for a response. Active object 2 sends a sequential message (1.1) to passive object 4 to accomplish some behavior before active object 2 responds to active object 1’s request, at which point active object 1 sends an asynchronous message (2) to active object 3. Note that all system operations are considered asynchronous. Furthermore, all messages between active objects must be either asynchronous or synchronous.
4.2.2.2 Communication and Synchronization Identification

The threads of control identified in Section 4.2.1 on page 42 are useful in determining the type of messaging taking place between objects. Several rules are derived for the receivers of messages. First, passive objects may receive only sequential and synchronous messages. Second, active objects may receive only asynchronous and synchronous messages.
Beyond these rules, the mutual exclusion semantics of the individual objects determined in Section 4.2.3 on page 46, must be examined to determine additional communication and synchronization information. If an operation is maximally sequential or mutual exclusion properties exist, then sequential messaging may not be used, and either synchronous or asynchronous messaging must be employed.

Finally, when synchronous messaging is used, the visibility graphs aid in determining the type of synchronous messaging necessary (i.e., synchronous, balking, or time-out). If a reference is exclusive, then balking is typically appropriate as no other object should be using the reference. If a balking message fails, then an error has occurred and appropriate measures may be taken. For shared references, the semantics of the sender must be examined to determine if it may block, and if so, for how long.

4.2.3 Mutual Exclusion Extensions

Mutual exclusion is not an issue in traditional Fusion as all operations are sequential and an object is limited to servicing one request to its entirety before servicing another. Extensions are needed to allow multiple threads of control within an object while maintaining the integrity of that object's attributes in the presence of multiple threads of control.

4.2.3.1 Mutual Exclusion Notation

The following notations are added to the class description diagrams in order to denote the mutual exclusion properties of an object's operations. Figure 4.3 on page 47 shows the extensions to the class description syntax marked in bold text. The first extension is a result of the discovery of threads of control from Section 4.2.1 on page 42. The concurrency properties of the object (i.e., either active or passive) are included in the class description.

The second extension is the designation of the object's operations as either maximally concurrent or maximally sequential. Maximally concurrent operations place no restrictions on the number of threads present. Maximally sequential operations are limited to one thread at any given point in time.
The final extension is the inclusion of mutual exclusion properties for the operations. Any operations which must run mutually exclusive of one another are noted as such.

Figure 4.4 on page 47 shows the new template for class descriptions.

```plaintext
class <ConProp> <ClassName> [isa <SuperClassNames>]
    // for each attribute
    [attribute] [Mutability]<a_name> :[Sharing][Binding]<Type>
    // for each method
    [method] [MethodMutEx] <m_name> <arglist>[:<Type>]
    // for each mutual exclusion property
    [MutEx] <m_name_list>
endclass
```

FIGURE 4.4 Extended Class Description Template
4.2.3.2 Mutual Exclusion Identification

Unfortunately, it is difficult to determine the mutual exclusion properties of an objects operations other than by examining the semantics of the operations of themselves. Each operation should be examined to determine what, if any, object resources (i.e., attributes) are being manipulated, either through modification or simple viewing. Any operation which modifies an object resource must be declared as maximally sequential. Furthermore, any “read” and “write” operations which access the same object resource must be declared mutually exclusive of one another. Note that any explicit create or remove operations included in the object are implicitly maximally sequential and mutually exclusive and need not be considered.

Figure 4.5 shows an example of how the extended class descriptions might be used to convey mutual exclusion properties for a simple buffer class. Note that the write operation updates a object resource and is declared as maximally sequential. The write and read operations both manipulate the same object resource and therefore must be declared as being mutually exclusive of one another.

```
class Passive buffer
  method MaxSeq write(item : item_type)
  method MaxCon read() : item_type
    Mutex(write, read)
endclass
```

FIGURE 4.5 Mutual Exclusion Example

4.2.4 Fusion Process Extensions

As mentioned previously, the concurrency modeling extensions should integrated into the existing Fusion model as tightly and as seamlessly as is possible. To this end, the extended Fusion process is summarized below. Additions to the process for concurrency modeling are added in bold text. These process changes, in addition to the previously mentioned notation and identification extensions, form the concurrency modeling extensions used in Chapter 6.
4.2.4.1 Object Interaction Graphs Process Extensions

The process for the creation of the object interaction graph is summarized as follows:

1. Identify relevant objects involved in the computation.

2. Establish the role of each object participating in the computation.
   - Identify controller.
   - Identify collaborators.

3. Decide on messages between objects.
   - **Identify asynchronous messages.**
   - **Identify synchronous messages with the aid of visibility graphs and class descriptions.**

4. Decide on thread of control for objects using system events and internal messaging as a guideline.

5. Record how the identified objects interact on an object interaction graph.

Verify the following:

1. Consistency with analysis models. Check that each of the classes in the system object model are represented in at least one object interaction graph.

2. Verification of functional effect. Check that the functional effect of each object interaction graph satisfies the specification of its system operation given in the operation model.

3. **Verify thread of control information:**
   - All objects receiving system events and internal asynchronous messaging are marked active.

4. **Verify message synchronization:**
   - All messages to active objects are either synchronous or asynchronous.
   - All messages to passive objects are either synchronous or sequential.
   - Passive objects may not send messages to active objects.
   - Methods with maximally sequential or mutual exclusion requirements from the class descriptions receive only synchronous or asynchronous messages.
4.2.4.2 Class Descriptions Process Extensions

Class descriptions derive their information from the system object model, object interaction graphs, and visibility graphs and record the following:

1. Methods and parameters from the object interaction graphs.
2. Data attributes from the system object model and data dictionary.
3. Object attributes from the visibility graph for the class.
4. Inheritance information from the inheritance graphs.
5. **Method concurrency and mutual exclusion properties derived from examination of method semantics.**

Verify the following:

1. All methods from the object interaction graphs are recorded.
2. All data attributes from the system object model are recorded.
3. All visibility references are recorded.
4. All inherited superclasses are recorded.
5. **All concurrency and mutual exclusion properties for the methods are recorded.** Verify that:
   — All “writers” of object resources are declared maximally sequential.
   — All “readers” and “writers” of the same object resource are declared as being mutually exclusive of one another.

4.2.5 Inheritance of Concurrency

As this thesis is concerned primarily with the analysis and high level or architectural design phases of Fusion rather than low level design and implementation issues, how the concurrency modeling constructs presented here are inherited between classes is not currently covered. Consequently, no extensions to Fusion’s inheritance graphs are provided. Suggestions for future research on this topic are included in Chapter 7.
This chapter provides an overview of Cellular Digital Packet Data (CDPD) as defined in [16]. CDPD is used in the following chapter as an example problem domain in which the previously proposed concurrency extensions to the Fusion Development Methodology are demonstrated. Briefly, CDPD is an emerging technology which provides connectionless packet data networking services to mobile hosts via the use of existing radio channels typically allocated for Advanced Mobile Phone Systems (AMPS) cellular service (i.e., traditional analog cellular service). As an example problem domain, CDPD is sufficiently "rich" to show the need for concurrency modeling typical of most distributed, multi-processor, and multi-processing telecommunications and data communications domains.

This overview attempts to provide the reader with a broad architectural view of the entire CDPD system. As the distinguishing feature of CDPD, the mobility management aspects present in a mobile packet data environment are also covered in greater detail. The reader is referred to [16] if more information regarding CDPD is desired.

5.1 OSI Network Model

The Architecture used in the construction of the CDPD Network follows the OSI model for layered protocols. This section gives a brief overview of the OSI network model and its terminology. It is assumed that the reader is familiar with the structure and functionality of the OSI layered model.

An OSI network, as shown in Figure 5.1 on page 52, consists of End Systems (ESs) interconnected to one another by a set of Intermediate Systems (ISs). In Internet terminology, ESs are "hosts" and ISs are "routers." Note that the individual networks are

---

1 This author wishes to acknowledge the copyright held by CDPD Forum Inc. on the CDPD System Specification and gratefully thank them for allowing the contents of the document to be used for academic purposes.
often combined into a larger network or internetwork. Each network often has its own administrative and routing policies and are called “administrative domains.”

![Diagram of an Internetwork of Administrative Domains](image)

**FIGURE 5.1 An Internetwork of Administrative Domains**

The purpose of a network, in simple terms, is to allow data to be transmitted to, and received from, equipment (i.e., ESs) that is attached to the network. ESs are the logical endpoints of these communications and become the source and/or destination network entities. It is important to note that each ES is identified by at least one globally distinct Network Entity Identifier (NEI).

ISs implement the network layer functionality as defined by ISO-7489. The network layer must find a path through a series of connected ISs until the desired ES is reached. ISs along the path must forward network layer packets in the appropriate direction. They also deal with route discovery and calculation, fragmentation, reassembly and congestion mitigation. The connectionless network service, sometimes referred to as datagram service, routes each packet through the network based solely on the destination address carried in the packet and knowledge of current network topology.
5.2 CDPD Network Model

The CDPD Network encompasses several important concepts above and beyond the standard OSI network. First, it is designed to interwork with existing data networks. Second, and most importantly, it is designed to deliver data to mobile hosts over wide geographic areas and makes a distinction between Mobile End Systems (M-ESs) and Fixed End Systems (F-ESs) for the purposes of mobility management. Finally, it uses existing cellular channels for air data link services. Each of these important aspects is dealt with in greater detail in the following overview sections with Section 5.3, "CDPD Mobility Management," on page 62 provided additional details on mobility management.

To aid on the discussion, several network points of reference are introduced and shown in Figure 5.2.

- The Air Link Interface or reference point (A) allows CDPD subscribers to use their M-ESs to access network service.
- The External Interface or reference point (E) allows external applications service providers to communicate with CDPD subscribers.
• The Inter-Service Provider Interface or reference point (I) allows support of CDPD Network services across all areas served by CDPD Service Provider domains.

5.2.1 Interface to Existing Networks

The CDPD Network is designed to operate as an extension of existing data communications networks through the (E) interface. This extension is provided through the definition of the CDPD network as a peer multi-protocol, connectionless network to existing data infrastructures. The CDPD Network provides data communications services consistent with the Connectionless Network Layer Service (CNLS) definition found in ISO-8348.

As a "multi-protocol" connectionless network, the CDPD Network is capable of providing network services in any of several different network protocols. Currently, Connectionless Network Protocol (CNLP), the standard OSI connectionless network protocol defined in ISO-8473, and Internet Protocol (IP), the network layer protocol of the Internet TCP/IP protocol suite defined in RFC-791, are supported. Figure 5.3 on page 55 shows the CDPD Network as an extension of existing data networks.
Extending the idea of Administrative Domains to CDPD Service Providers, Figure 5.1 on page 52 and Figure 5.3 on page 55 may be combined and re-drawn as shown in Figure 5.4 on page 56 to encompass both Mobile and Fixed End Systems, the (A) Interface connecting the M-ESs to CDPD Service Providers, the (I) Interface connecting the CDPD Service Providers, and the (E) Interface connecting the CDPD Network to the existing external network.
5.2.2 Mobility Management Overview

The very nature of cellular communications requires the CDPD Network to support a wide variety of mobility issues including the reliable and efficient delivery of packets between end systems during the transfer of an M-ES between two “cells” operated by the same CDPD service provider as well as the delivery of packets to M-ESs roaming outside of their “home” service provider’s coverage area.

Mobility management in the CDPD Network requires the introduction of several new elements to the network model. First, the CDPD network distinguishes between ISs aware of M-ESs mobility (Mobile Data Intermediate Systems or MD-ISs) and those ISs which are not aware of mobility. Second, the CDPD Network introduces a new entity known as the Mobile Data Base Station (MDBS) which provides layer 2 data link or media access relay functions for a set of radio channels serving a particular cell. Finally, as mentioned briefly before, the CDPD Network distinguishes between Mobile and Fixed End Systems (M-ESs and F-ESs, respectively). Each of these mobility management issues
are covered in greater detail in the following sections. Figure 5.5 shows the updated network reference model for the CDPD Network including these new entities.

![CDPD Network Reference Model](image)

**FIGURE 5.5 CDPD Network Reference Model**

5.2.2.1 Mobile Data Intermediate System (MD-IS)

Mobile Data Intermediate Systems (MD-ISs) perform routing functions based on additional knowledge of the current location of M-ESs. MD-ISs are the only network relay systems that have any knowledge of mobility and operate a CDPD-specific Mobile Network Location Protocol (MNLP) to exchange location information. The MD-IS performs two distinct mobility routing functions, the Mobile Home Function (MHF) and Mobile Serving Function (MSF), which cooperate to provide location-independent network service.

Every M-ES is logically a member of a fixed home area. The home area provides the anchor or mobility-independent routing destination area for ISs and ESs that are not mobile-aware. The Mobile Home Function (MHF) in the home area MD-IS operates a packet forwarding service in the forward direction only (i.e., toward the M-ES), and maintains a database of the current serving area for each of its M-ESs. The mobile forwarding function is based on the principle of encapsulating M-ES addressed packets,
and forwarding them to the Mobile Serving Function (MSF) in each serving area the M-ES visits.

The Mobile Serving Function (MSF) of an MD-IS handles the routing of packets for all visiting M-ESs in its serving area. When an M-ES registers for network access in an MD-IS serving area, the MSF notifies the home MD-IS of its current location. The serving MD-IS layer management functions cooperate with the network support service applications for accounting of the use of the network services by the M-ES. On the air link side, an MD-IS performs the functions of a mobile-specific ES-IS router (i.e., it routes data for M-ESs within its area toward the current subnetwork point of attachment based on local knowledge of the subscriber’s current cell).

Both the MHF and MSF services are covered in more detail in Section 5.3, “CDPD Mobility Management,” on page 62.

5.2.2.2 Mobile Data Base Station (MDBS)

As mentioned previously, the Mobile Data Base Station (MDBS) provides layer 2 data link relay functions for a set of radio channels serving a cell. The MDBS is responsible for detailed control of the radio interface, such as radio channel allocation, inter-operation with cellular voice channel usage, and radio media access control. Each cell requires a logical MDBS function, with cells containing multiple CDPD channel streams requiring a logical MDBS function for each stream. A channel stream is a bidirectional communications path between an MDBS and a group of M-ESs, using a single RF channel at a time. More information on the layer 2 relay functions of the MDBS are found in Section 5.2.3, “Air Data Link,” on page 59.

5.2.2.3 Mobile End Systems (M-ESs)

The M-ES is the means by which CDPD Network subscribers gain access to wireless communications. The M-ES may be physically mobile or stationary, but it is considered as always being potentially mobile. By definition, the physical location of an M-ES may change with time, but continuous network access is maintained. To accomplish this transparent movement, the M-ES and cooperating MDBSs and MD-ISs perform
additional operations in the network sublayer and below allowing the M-ES to move from cell to cell or network to network (i.e., change its subnetwork point of attachment) while having no observable effect on the higher layer functions.

Networks traditionally route information based on destination addresses stored in the respective datagrams. Traditional network connectivity and routing functions cannot be used in the CDPD Network because the location of M-ESs, and the route to reach them, cannot be determined from their network addresses. Additional M-ES mobility support is required from the CDPD Network to track the M-ESs current subnetwork point of attachment (SNPA) and route Network Protocol Data Units (NPDUs) accordingly. Also, radio resource management functions are required which are capable of discovering and maintaining suitable SNPAs.

In summary, each M-ES is aware of its location (i.e., a cell), based on the channel stream it is currently using. When the M-ES moves to another cell, it notifies the network by indicating a change to the channel stream used in the new cell.

5.2.3 Air Data Link

The use of existing cellular channels for the air data link service involves the introduction of the CDPD Air Link Interface. CDPD uses several new procedures and protocols to allow the efficient use of cellular channels for data communications. These include the Mobile Data Link Protocol (MDLP) and CDPD Media Access Control (MAC) protocols. The actual radio channel constitutes the physical layer of the CDPD system. The physical layer accepts a sequence of bits from the MAC layer and transforms them into a modulated waveform for transmission to the M-ES. Conversely, the physical layer also receives a modulated waveform from the M-ES which is translates into a sequence of received bits for delivery to the MAC layer.

The MAC layer is part of the CDPD data link layer which is divided into logical link control functions, handled by MDLP, and media access control functions, handled by CDPD MAC. The purpose of the MAC sublayer is to convey information between MDLP entities on the MDBS and M-ES across the CDPD physical airlink interface. All MDLP data units are transmitted in frames that are delimited by MAC layer functions. These
frames are further block encoded for error protection and transmitted in bit serial form on the channel stream. The MAC layer is defined to allow multiple M-ESs to share the common aircircuit interface. The M-ES sends messages in bit serial form when it senses that the medium is idle. If collisions are detected, the M-ES remains quite for a random amount of time before attempting another transmission.

The purpose of MDLP is to convey information between Network Layer entities across the CDPD Airlink (A) interface. MDLP provides for the provision of one or more logical data link connections on a channel stream. The discrimination between data link connections is by means of an address known as the Temporary Equipment Identifier (TEI) contained in each frame. MDLP also provides sequence control, error recovery, and flow control.

From an administrative point of view, MDLP allows M-ESs to request that the MD-IS assign a TEI that the M-ES can use in its subsequent communications. MDLP also allows an MD-IS to remove a previously assigned TEI and to verify whether or not a TEI value is in use. Figure 5.6 shows a scenario diagram showing the most basic operation of TEI assignment.

![Figure 5.6 TEI Assignment](image-url)
Figure 5.7 shows a scenario diagram showing the most basic operation of TEI removal.1

Figure 5.8 shows scenario diagram showing the most basic operation of TEI Checking.

In summary, Figure 5.9 on page 62, depicts the Airlink protocols along with the remainder of the protocol stacks employed in the CDPD Network.
5.3 CDPD Mobility Management

The most unique aspect of the CDPD Network is that M-ESs may change their subnetwork point of attachment (SNPA) at will, thus traditional forms of network connectivity and routing cannot be used because the M-ESs' location and route to reach them cannot be deduced from their network address. The CDPD Network introduces the function of "mobility management" to administer the maintenance of a location information database and routing of Network Protocol Data Units (NPDUs) based on this information. A closely related function, not covered here in great detail, is that of radio resource management which is concerned with the discovery, selection, and use of the most optimal radio channels available to the M-ES.

The CDPD Network provides transparent mobile interworking meaning that the mobility of M-ESs is completely transparent to upper layer protocols. The mobility information is managed and constrained within MD-ISs so that routing and relaying are
transparent to all other entities internal and external to the CDPD Network. Support for transparent mobility is provided at both the Data Link Layer and Network Layer in the CDPD Network. Data Link Layer support for mobility provides Network Layer transparency over a group of cells forming the service coverage area of an MD-IS. Likewise, Network Layer support for mobility provides Transport Layer transparency to mobility over the entire CDPD Network.

The key concept behind CDPD mobile interworking is that every M-ES is logically a member of a fixed home area. The home area provides the anchor or mobility-independent routing destination area for ISs and F-ESs that are not mobile aware. The CDPD mobility functions provide for rerouting and forwarding NPDUs from the home area to the current location of the M-ES. The MD-IS encapsulates several services which cooperate to provide this location-independent network service.

5.3.1 Mobile Home Function (MHF)

The MD-ISs Mobile Home Function (MHF) consists of two services. First, a Location Directory Service which maintains an information base of the current serving area for each of its homed M-ESs. The CDPD-specific Mobile Network Location Protocol used to update this Location Directory is covered in more detail in Section 5.3.4, “Mobile Network Location Protocol (MNLP),” on page 65.

Second, the Redirection and Forwarding Service based on the principle of encapsulating M-ES addressed packets and forwarding them to the Mobile Serving Function (MSF), see below, in the serving area in which the M-ES is currently visiting. In the forward direction (i.e., packets destined for an M-ES), packets are routed by traditional means first to the MD-IS in the M-ESs home area, then encapsulated and tunneled to the MD-IS in the current serving area, and finally routed to the M-ES at its current cell location.

Note that packets originating from an M-ES are routed directly to their destination by traditional means. There is no requirement for packets in the reverse direction to transit the home MD-IS.
5.3.2 Mobile Serving Function (MSF)

The Mobile Serving Function (MSF) of an MD-IS handles the routing of packets for all visiting M-ESs in its serving area. When an M-ES registers for network access via the Mobile Network Registration Protocol (MNRP), see Section 5.3.5, “Mobile Network Registration Protocol (MNRP),” on page 74, the MSF notifies the home MD-IS of the M-ESs current location via the MNLP, see Section 5.3.4, “Mobile Network Location Protocol (MNLP),” on page 65. The MSF consists of two services. First, the Registration Directory Service which maintains an information base of the M-ESs currently registered in the service area. Second, the Readdress Service which decapsulates forwarded NPDUs from the MHF and routes them to the correct channel stream in cell.

Note that the serving MD-IS also cooperates with network support service applications for authentication, authorization, and accounting of the use of network services by the M-ES, all of which are beyond the scope of this overview.

5.3.3 Redirection and Forwarding

Figure 5.10 on page 65 provides a pictorial summary of the Redirection, Forwarding, and Readdressing services provided by the MHF and MSF. This section also provides a walk-through of NPDU flow through the CDPD Network.

First, the Redirection service at the home MD-IS participates in the standard Network Layer routing protocols such that it advertises direct Network Layer reachability to Network Service Access Point (NSAP) addresses corresponding to its home area M-ESs. NPDUs addresses to an M-ES will, therefore, be routed and relayed by traditional means to the Redirection Service associated with the destination M-ES.

When the Redirection Service receives an NPDU addressed to one of its home area M-ESs, it queries the Location Directory to obtain a mapping between the home area NSAP address and the current forwarding address. The Redirection Service encapsulates the original NPDU in a new NPDU addressed to its Forwarding Address. The encapsulated NPDU is then routed by traditional means to the Readdress Service at the serving MD-IS where the M-ES is currently located.
The Readdress Service at the serving MD-IS then receives the NPDU and decapsulates it, restoring the original NPDU addressed to the destination M-ES. The original NPDU is then routed to the current cell of the M-ES.

When an M-ES originates an NPDU, the NPDU is routed by traditional means to its destination. There is no need to redirect NPDU's through the originator's home MD-IS. If the destination happens to be another M-ES, the NPDU will be routed to the M-ESs home area and handled as above.

5.3.4 Mobile Network Location Protocol (MNLP)

The Mobile Network Location Protocol (MNLP) provides several services. First, it enables notification to the MD-IS of the current location of NEIs associated with an M-ES. Second, it allows the transportation of data origin authentication information about M-ESs and their NEIs. Third, it allows confirmation from the home MD-IS to the serving MD-IS of the willingness and ability to provide network forwarding services on behalf of an M-ES at its current location. Finally, it enables the forwarding of NPDU's addressed to an M-ES from the home MD-IS to the current serving MD-IS. All of these services may be summarized in two functions: the Location Update function and the Forward NPDU function described below.
Before describing these two functions, MNLP requires several parameters and information bases which will be described first.

5.3.4.1 MNLP Parameters

MNLP parameters consist of two NSAP addresses and one timer value. Two NSAP addresses are used to designate the Service Access Points for 1) transfer of location update information between MD-ISs and 2) transfer of NPDUs between the MD-ISs by the redirection and forwarding service.

MNLP also employs one additional parameter known as the Holding Timer. The value of the Holding Timer is set by the home MD-IS and indicates how long location information is retained by the Location Directory, see Section 5.3.4.2, “Forwarding Information Base,” on page 66. Old location information is discarded after the Holding Timer expires to ensure correct operation of the protocol.

5.3.4.2 Forwarding Information Base

The Forwarding Information Base used by MNLP consists of three separate directories of information. First, the Location Directory used by the MHF in locating M-ESs consists of a set of tuples containing the M-ES NEI, M-ES Registration Sequence Count, and MD-IS Readdress Server Forwarding Address. Second, the Registration directory used by the MSF consists of a set of tuples consisting of M-ES NEI, M-ES Registration Sequence Count, and M-ES SNPA Address.

Finally, the Home Domain Directory used by the serving MD-IS to determine the NSAP address of the Location Update service in the home MD-IS for a given M-ES consists of a set of tuples containing Area Address, Address Mask, and Location Update Service NSAP Address. The Address Mask is used to mask the M-ESs NEI with a masked bit pattern matching that in the Area Address indicating the NEIs home MD-IS address in the Location Service NSAP Address field.
5.3.4.3 Location Update Function

The Location Update Function is used by MD-ISs to exchange information about the current location and reachability of M-ES NEIs. The Location Update Function encompasses the Report Location Function, the Record Location Function, the Flush Registration Information Function, the Flush Location Information Function, and the Query Location Function. Each one of these “sub-functions” is dealt with in detail below.

The Report Location Function allows a serving MD-IS to update the home MD-ISs Location Directory. Upon receipt of an End System Hello (ESH) Protocol Data Unit (PDU), see Section 5.3.5, “Mobile Network Registration Protocol (MNRP),” on page 74, the Report Location Function first updates the Registration Directory from configuration information contained in the ESH PDU. The Report Location Function then begins construction of a Redirect Request (RDR) PDU. The RDR PDU contains the Source Network Address (i.e., the M-ES NEI), Registration Sequence Count, Location Info parameter, and all other optional parameters included in the ESH PDU. The RDR PDU also contains the NSAP address of the serving MD-ISs Forward NPDU service. The RDR PDU is then addressed and transmitted to the Location Update service found in the Home Domain Directory for this M-ES NEI, see Section 5.3.4.2, “Forwarding Information Base,” on page 66. The Report Location Function then waits for the receipt of a Redirect Confirm (RDC) PDU. Upon receipt of the RDC PDU, the Report Location Function first checks to ensure that the Registration Sequence Count contained in the RDC matches the one stored in the Registration Directory. If they do not match, the RDC PDU is discarded and no other action is taken. Otherwise, the Report Location Function begins construction of the MD-IS Hello Confirm (ISC) PDU to the M-ES NEI containing information extracted from the RDC PDU, including the Authentication Update parameter, Result Code parameter, and Configuration Timer parameter. Finally the ISC PDU is transmitted to the M-ES NEI and the Report Location Function examines the Result Code parameter returned in the RDC PDU, removing the M-ES NEIs Registration Directory entry if the code indicates that service has been denied by the home MD-IS.

Figure 5.11 on page 68 shows a scenario diagram for the most basic Report Location Functions.
a. Upon receipt of an MNRP ESH, the Report Location Function constructs, routes, and transmits an MNLP RDR to the Location Update Service address of the Home MD-IS associated with the M-ES identified in the ESH.

b. Upon receipt of an MNLP RDC, the Report Location Function constructs and transmits an MNRP ISC to the M-ES.

FIGURE 5.11 Report Location Function

The Record Location Function updates a home MD-IS’s Location Directory based on information received from a serving MD-ISs Report Location Function. Upon receipt of the RDR PDU, the Record Location Function extracts the location information and compares it with information stored in the Location Directory. If the Source Network Address (i.e., the M-ES NEI) does not belong to this Location Directory, the RDR is rerouted and forwarded by querying the Home Domain Directory and no other action is taken. Otherwise the Record Location Function makes a check of the Registration Sequence Count included in the RDR PDU. If the Registration Sequence Count indicates an out-of-sequence registration attempt, then a Redirect Flush (RDF) PDU is constructed and sent to the originator of the RDR without the Location Directory being updated. Otherwise, any necessary authentication and access control procedures are performed based on the NEI, its authentication credentials, and its location information parameter.
The authentication process generates a Result Code and may generate an update to the M-ES authentication credentials. Once authentication is complete, the Record Location Function then updates the Location Directory, restarts the Holding Timer, and constructs a Redirect Confirm (RDC) PDU containing the Registration Sequence Counter from the RDR, the Home Information parameter, and Result Code and transmits this PDU to the Location Update Service NSAP address. Finally, if the information in the RDR indicates a change of serving area (i.e., the M-ES has moved to a new MD-IS) then a Redirect Flush (RDF) is constructed and transmitted to the previous Location Update Service NSAP address associated with the M-ES. Figure 5.12 shows a scenario diagram for the most basic Record Location Functions.

![Diagram](image)

**FIGURE 5.12 Record Location Function**

The Flush Registration Information Function has two responsibilities pertaining to the cleanup of the Registration Directory. Upon receipt of the Redirect Flush (RDF) PDU, the Flush Registration Information Function removes the corresponding entry from the
Registration Directory. Also, upon receipt of an End System Bye (ESB) PDU, see Section 5.3.5, “Mobile Network Registration Protocol (MNRP),” on page 74, the Flush Registration Information Function constructs a Redirect Expiry (RDE) PDU containing the Source Network Address (i.e., M-ES NEI) from the ESB and the current Registration Sequence Count from the Registration Directory. The RDE is addressed to the Location Update Service NSAP address determined by querying the Home Domain Directory for the M-ES NEI identified by the ESB. The Flush Registration Information Function then transmits the RDE and removes the Registration Directory entry corresponding to the M-ES NEI contained in the ESB. Figure 5.13 shows a scenario diagram for the most basic Flush Registration Information Functions.

![Diagram of Flush Registration Information Function]

a. Upon receipt of an MNLP RDF, the serving MD-IS removes the corresponding registration directory.

b. Upon receipt of an MNRP ESB, the serving MD-IS removes the correspond registration directory and constructs, routes, and transmits an MNLP RDE to the Location Update Service address of the M-ES’s home MD-IS.

FIGURE 5.13 Flush Registration Information Function

The Flush Location Information Function also has two responsibilities pertaining to the cleanup of the Location Directory. Upon expiry of the Holding Timer entry, the
Flush Location Information Function removes the corresponding entry from the Location Directory. Also, upon receipt of the RDE PDU, the Flush Location Information Function first compares the Registration Sequence Count in the RDE to the value stored in the Location Directory. If the two values do not match, the PDU is discarded and no other action is taken. Otherwise, the Location Directory entry corresponding to the M-ES NEI contained in the RDE PDU is removed. Figure 5.14 shows a scenario diagram for the most basic Flush Location Information Functions.

![Diagram](image)

**FIGURE 5.14 Flush Location Information Function**

The Query Location Function allows a home MD-IS to validate the Location Directory information for a given NEI. The Query Location Function may also be invoked to initiate a re-validation of authentication information of the user of the NEI. When the home MD-IS wishes invoke the Query Location Function, the home MD-IS constructs and transmits a Redirect Query (RDQ) PDU to the serving MD-ISs Location Update Service NSAP address. Upon receipt of an RDQ, the Query Location Function first queries the Registration Directory for the M-ES NEI. If no entry is found, an RDE PDU is
constructed and returned to the originator. If the entry is found, an End System Query (ESQ) PDU, see Section 5.3.5, “Mobile Network Registration Protocol (MNRP),” on page 74, is constructed and transmitted to the M-ES NEI. Figure 5.15 shows a scenario diagram for the most basic Query Location Functions.

a. The home MD-IS constructs and transmits an MNLP RDQ.
b. Upon receipt of an MNLP RDQ, the serving MD-IS either returns and MNLP RDE if no registration directory entry is found or constructs and transmits and MNRP ESQ to the corresponding M-ES.

FIGURE 5.15 Query Location Function

5.3.4.4 Forward NPDU Function

The Forward NPDU Function is used by MD-ISs to forward NPDUs destined for M-ESs from the home MD-IS to the current serving MD-IS. The Forward NPDU Function consists of the Redirection Function and the Readdress Function. Each one of these “sub-functions” is dealt with in detail below.

The Redirection Function is invoked when the MHF receives an NPDU for an M-ES not currently served by its home MD-IS (i.e., no entry in the Registration Directory). When the MHF receives such an NPDU, the Redirection Function is used to first query the Location Directory. If an entry is found then the Redirection Function encapsulates the NPDU in a new NPDU sends the packet to the NSAP address associated
with the current MSF forwarding service as determined from the Location Directory. If no Location Directory is found, then error reporting procedures are invoked and the original NPDU is discarded. Figure 5.16 shows a scenario diagram for the most basic Redirection Functions.

a. MHF receives an NPDU and encapsulates it in a new NPDU addressed to the MSF Forwarding Service address currently associated with the M-ES.

FIGURE 5.16 Redirection Function

The Readdress Function is invoked when the MSF receives an NPDU via its forwarding service NSAP. The Readdress Function first decapsulates the original NPDU and queries the Registration Directory based on the M-ES NEI stored in the NPDU. If the Registration Directory entry exists then the Readdress Function relays the NPDU to the current M-ES NEI. Otherwise, an RDE PDU is constructed and transmitted as per Section 5.3.4.3, “Location Update Function,” on page 67, error reporting procedures are invoked, and the original NPDU is discarded. Figure 5.17 on page 74 shows a scenario diagram for the most basic Readdress Functions.
NPDU

a. MSF receives an NPDU on its Forwarding Service address, decapsulates it and forwards it to the subnetwork address associated with the M-ES.

FIGURE 5.17 Readdress Function

5.3.5 Mobile Network Registration Protocol (MNRP)

The Mobile Network Registration Protocol (MNRP) provides configuration information about M-ESs to MD-ISs. An MD-IS is informed of the NEIs supported by each M-ES and the subnetwork point of attachment (SNPA) address of the M-ES. In other words, a serving MD-IS obtains information that allows it to associate destination network addresses to specific channel streams and to specific data link connections on the channel stream. Once the MD-IS obtains this information, reachability information and routing metrics concerning these NEIs can be disseminated to other MD-ISs for the purpose of calculating routes to/from each M-ES on a subnetwork. Note that MNRP logically performs the same functions as OSI's ES-to-IS Routing Exchange Protocol specified in ISO-9542.

MNRP provides several services. First, it enables notification of reachability and registration of NEIs associated with an M-ES to a serving MD-IS. Second, it allows the transportation of data origin authentication information about M-ESs and their NEIs. Third, it allows confirmation by an MD-IS of its willingness and ability to provide network routing services to an M-ES. Finally, it allows deregistration of an NEI from the serving MD-IS.
Before describing these functions, MNRP requires several parameters which will be described first.

5.3.5.1 MNRP Parameters

The Configuration Timer is a timer local to the M-ES which assists it in performing the Report Configuration Function. This timer determines how often an M-ES reports its availability to the serving MD-IS. An MD-IS does not maintain a Configuration Timer.

The Holding Timer is set by the MD-IS and governs how long configuration information is retained by the MD-IS. Old configuration information is discarded after the Holding Timer expires to ensure correct operation of the protocol.

The Response Timer is a timer local to the M-ES which assists it in performing the Report Configuration Function. This timer determines how long an M-ES waits for confirmation after reporting its configuration information. An MD-IS does not maintain a Response Timer.

The Registration Sequence Counter is a counter maintained by each M-ES that assists it in performing the Report Configuration Function. The Registration Sequence Counter is incremented by one on each initial establishment of the point-to-point data link connection between an M-ES and MD-IS. A common Registration Sequence Counter is used by all NEIs present at an M-ES.

5.3.5.2 Report Configuration Function

The Report Configuration Function is used by M-ESs to inform the current serving MD-IS of their reachability and current subnetwork address. To register an NEI, the M-ES constructs and transmits an End System Hello (ESH) PDU containing the authentication credentials of the NEI on the previously established data link connection between an M-ES and MD-IS. This function is invoked (1) after the initial establishment of the point-to-point data link connection between an M-ES and MD-IS, (2) following the expiration of the M-ESs Configuration or Response Timers, and (3) during activation of an NEI subsequent to initial data link establishment.
To deregister an NEI, the M-ES constructs and transmits an End System Bye (ESB) PDU on the previously established data link connection to inform the MD-IS about an NEI that is no longer active. This function is invoked when (1) the M-ES determines that the NEI is no longer active, (2) upon the receipt of an NPDU via the point-to-point data link connection addressed to an unknown or inactive NEI, and (3) prior to the release of the underlying data link connect.

5.3.5.3 Record Configuration Function

The Record Configuration Function receives End System Hello (ESH), End System Byte (ESB), and MD-IS Hello Confirm (ISC) PDUs, extracts the configuration information, and updates the routing information base. On receipt of an ESH PDU, the serving MD-IS stores the \{NEI, SPNA\} pairs in its local information base. The serving MD-IS also performs the Location Update procedures described in Section 5.3.4.3, "Location Update Function," on page 67. Upon completion of the location update procedure, the serving MD-IS then constructs an ISC PDU and transmits it on the SNPA address stored against the NEI. On receipt of an ESB PDU, an MD-IS removes the NEI configuration corresponding to the \{NEI, SPNA\} pair from its information base. The MD-IS again performs the Location Update procedures described above.

On receipt of an ISC PDU, the M-ES stops its Response Timer and extracts the Result Code, Configuration Information, and Authentication Information. If the Result Code indicates that network service has been denied, the M-ES is then prohibited from sending additional NPDUs with the same source address NEI.

Figure 5.18 on page 77 shows a scenario diagram for the most basic Record Configuration Functions.
a. Upon receipt of an MNRP ESH, the Record Configuration Function extracts and store configuration and \{NEI, SPNA\} pairs in its information base before initiating location update procedures.

b. Upon completion of the location update procedures, the Record Configuration Function then constructs and transmits and MNRP ESB to M-ES's current subnetwork point of attachment.

c. Upon receipt of an MNRP ESB, the Record Configuration Function removes the corresponding \{NEI, SNPA\} pair from its information base before initiating location update procedures.

FIGURE 5.18 Record Configuration Function

5.3.5.4 Flush Old Configuration Function

The Flush Old Configuration Function is executed to remove configuration entries in the local information base for which the Holding Timer has expired. When the Holding Timer for an MD-IS expires, this function removes the corresponding entry from the information base. The Flush Old Configuration Function is also executed when a
subnetwork service provider deactivates a local SNPA. When the SNPA is either disabled or deactivated, all configuration information for both M-ESs and MD-ISs associated with that SNPA is removed. The MD-IS also performs the Location Update procedures described in Section 5.3.4.3, “Location Update Function,” on page 67, at this time. Figure 5.19 shows a scenario diagram for the most basic Flush Old Configuration Functions.

a. The Flush Old Configuration Function removes the \{NEI, SNPA\} pair upon expiry of the Serving MD-IS’s Holding Timer before initiating Location Update procedures.

b. The Flush Old Configuration Function removes the \{NEI, SNPA\} pair upon deactivation of an M-ES’s SNPA before initiating Location Update procedures.

FIGURE 5.19 Flush Old Configuration

5.3.5.5 Query Configuration Function

The Query Configuration Function allows an MD-IS to verify that an NEI is present and active on a given data link connection and that the M-ES that is using the NEI can provide valid authentication information. The Query Configuration Function is invoked as part of the MNLP Query Location Function. Upon receipt of an End System Query (ESQ) PDU, the M-ES first determines if the NEI is present and active at the M-ES.

78
If so, the M-ES constructs and transmits an ESH PDU. If the NEI is either not present or not currently active, the M-ES simply ignores the ESQ PDU. Figure 5.20 shows a scenario diagram for the most basic Query Configuration Functions.

FIGURE 5.20 Query Configuration Function
CHAPTER 6
CASE STUDY

This chapter contains a case study using Cellular Digital Packet Data (CDPD) as the problem domain for exploring the effectiveness of the concurrency modeling extensions to Fusion proposed in Chapter 4. Due to the complexity of the CDPD problem domain, this case study focuses primarily on the mobility management areas of CDPD. Special detail is given to the Mobile Serving Function (MSF) and Mobile Home Function (MHF) and their use of the Mobile Network Registration Protocol (MNRP) and Mobile Network Location Protocol (MNLP) in mobility management. The reader will notice that the scope of case study narrows progressively through the analysis and design phases. The initial phases of Fusion analysis covered in Section 6.1 on page 80, provide a broad view of the CDPD system as a whole. The analysis then narrows its focus in the latter stages to mobility management issues. This trend continues through Section 6.2 on page 109, with the final output of the Fusion process being concerned exclusively with mobility management. This is done without compromising the goal of the case study which is to examine the effectiveness of the concurrency modeling extensions to Fusion proposed in Chapter 4. There exists ample concurrency issues within the narrowed problem domain to adequately explore the concurrency extensions of Chapter 4.

6.1 Analysis

As stated previously, Fusion analysis focuses primarily on the problem domain and it's externally visible behavior. Analysis begins with the object model used to capture classes, attributes, and relationships in the system. The life-cycle model then defines the allowable sequences of system operations. Finally, the operational model defines the semantics of each system operation.

6.1.1 Object Model

The following Entity Relationship (ER) diagrams and accompanying data dictionary tables identify the static structure of the CDPD system. Note that the entire
object model can not be represented by a single ER diagram and is thus broken down into several sub-diagrams. The complete object model is considered to be the union of all the sub-diagrams.

Figure 6.1 on page 82 depicts the object model for the major protocol classes in the MD-IS, MDBS, and M-ES and how these protocol classes are logically connected to one another. The layer 1 physical media between the M-ES and MDBS (i.e., radio transceivers), is logically connected by a Gaussian Filtered Minimum Shift Key (GMSK) modulated RF channel. As part of the lower layer 2 connection, the MAC protocols in the M-ES and MDBS are logically connected via a channel stream. Finally, the remainder of the layer 2 connection is shown by the logical connection of the two MDLP entities and an MDLP connection. Note that each MDBS may contain one or more physical and MAC entities which connect to zero or more corresponding entities in the M-ES. The MD-IS on the other hand, contains only one MDLP entity which creates zero or more MDLP point-to-point connections as it connects to either zero or one MDLP entity in each M-ES.
Table 6.1 on page 83 depicts the data dictionary for entities in the “CDPD Object Model: Protocol Connections” ER diagram.
### TABLE 6.1 Protocol Connection Data Dictionary

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-ES</td>
<td>class</td>
<td>Mobile End System.</td>
</tr>
<tr>
<td>Physical Protocol</td>
<td>class</td>
<td>Layer concerned with the actual transmission of data across a physical medium, in this case Advanced Mobile Phone System (AMPS).</td>
</tr>
<tr>
<td>Connects</td>
<td>relation</td>
<td>RF Channel logically connects the two Physical Protocols, one in the M-ES and one in the MDBS.</td>
</tr>
<tr>
<td>RF Channel</td>
<td>class</td>
<td>Portion of the electromagnetic spectrum defined by a central frequency and channel bandwidth.</td>
</tr>
<tr>
<td>MAC Protocol</td>
<td>class</td>
<td>Layer concerned with the control of access to a medium that is shared between two or more entities, in this case M-ESs.</td>
</tr>
<tr>
<td>Connects</td>
<td>relation</td>
<td>Channel Stream logically connects two MAC Protocols, one in the M-ES and one in the MDBS.</td>
</tr>
<tr>
<td>Channel Stream</td>
<td>class</td>
<td>A shared digital communications channel between and MDBS and a set of M-ESs considered as a logical concept, separate from the frequency of the RF channel used to implement the channel at any given time.</td>
</tr>
<tr>
<td>MDLP</td>
<td>class</td>
<td>Mobile Data Link Protocol. The link layer protocol used in CDPD.</td>
</tr>
<tr>
<td>Connects</td>
<td>relation</td>
<td>MDLP connection logically connects two MDLP entities, one in the M-ES and one in the MD-IS.</td>
</tr>
<tr>
<td>MDLP Connection</td>
<td>class</td>
<td>Connection between an M-ES and MD-IS.</td>
</tr>
<tr>
<td>MDBS</td>
<td>class</td>
<td>Mobile Data Base Station. Manages and accesses the radio interface from the network side. Relays and transmits packets sent from the MD-IS.</td>
</tr>
<tr>
<td>MD-IS</td>
<td>class</td>
<td>Mobile Data Intermediate System. The CDPD network element which performs routing functions based on knowledge of the current location of the M-ES.</td>
</tr>
</tbody>
</table>

Figure 6.2 on page 84 depicts the various entities in CDPD related to the cellular aspect of the system. The upper layer 2 MDLP connection makes use of zero or more
channel streams which in turn makes use of one or more RF channels when communicating with the M-ES. Each MDBS is in control of one or more cells which use the RF channel for communication with the M-ES. Note that each cell may be adjacent to zero or more other cells in the MD-IS's coverage area.

Table 6.2 on page 85 depicts the data dictionary for entities in the “CDPD Object Model: Cellular Entities” ER diagram.
### TABLE 6.2 Cellular Entities Data Dictionary

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDLP Connection</td>
<td>class</td>
<td>Connection between an M-ES and MD-IS.</td>
</tr>
<tr>
<td>uses</td>
<td>relationship</td>
<td>MDLP Connection uses a single Channel Stream.</td>
</tr>
<tr>
<td>Channel Stream</td>
<td>class</td>
<td>A shared digital communications channel between and MDBS and a set of M-ESs considered as a logical concept, separate from the frequency of the RF channel used to implement the channel at any given time.</td>
</tr>
<tr>
<td>uses</td>
<td>relationship</td>
<td>A Channel Stream may use one or more RF Channels at different times.</td>
</tr>
<tr>
<td>RF Channel</td>
<td>class</td>
<td>Portion of the electromagnetic spectrum defined by a central frequency and channel bandwidth.</td>
</tr>
<tr>
<td>MDBS</td>
<td>class</td>
<td>Mobile Data Base Station. Manages and accesses the radio interface from the network side. Relays and transmits packets sent from the MD-IS.</td>
</tr>
<tr>
<td>controls</td>
<td>relationship</td>
<td>MDBS controls one or more cells.</td>
</tr>
<tr>
<td>Cell</td>
<td>class</td>
<td>The region in which RF transmissions from one fixed transmission site can be received at acceptable levels of signal strength.</td>
</tr>
<tr>
<td>adjoins</td>
<td>relationship</td>
<td>Each Cell may be adjacent or adjoin zero or more Cells.</td>
</tr>
</tbody>
</table>

Figure 6.3 on page 86 depicts the relationships between the various physical entities of the CDPD system. Each MD-IS uses one or more IS’s to connect to either the CDPD or external existing packet networks. Each MD-IS is also in control of one or more MDBSs. Each MD-IS is adjacent to zero or more other MD-IS’s. Similarly, each MDBS is adjacent to zero or more other MDBSs. Finally, each MD-IS contains attributes representing its Location Update Service (LUS) NEI and Forwarding Service (FS) NEI as well as values for the various holding timers.
Table 6.3 on page 87 depicts the data dictionary for entities in the “CDPD Object Model: Physical Entities” ER diagram.
TABLE 6.3 Physical Entities Data Dictionary

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>class</td>
<td>Intermediate System. A node that is connected to more than one subnetwork with a primary role of forwarding data from one subnetwork to another.</td>
</tr>
<tr>
<td>uses</td>
<td>relationship</td>
<td>MD-IS uses one or more ISs to communicate with existing packet data networks.</td>
</tr>
<tr>
<td>MD-IS</td>
<td>class</td>
<td>Mobile Data Intermediate System. The CDPD network element and performs routing functions based on knowledge of the current location of the M-ES.</td>
</tr>
<tr>
<td>fs</td>
<td>attribute</td>
<td>Network Entity Identifier (address) for the forwarding service.</td>
</tr>
<tr>
<td>lus</td>
<td>attribute</td>
<td>Network Entity Identifier (address) for the location update service.</td>
</tr>
<tr>
<td>MSF holding interval</td>
<td>attribute</td>
<td>Interval in which registration directory information is to be maintained.</td>
</tr>
<tr>
<td>MHF holding interval</td>
<td>attribute</td>
<td>Interval in which location directory information is to be maintained.</td>
</tr>
<tr>
<td>MNRP configuration interval</td>
<td>attribute</td>
<td>Interval in which an M-ES reports its availability to an MD-IS.</td>
</tr>
<tr>
<td>adjoins</td>
<td>relationship</td>
<td>The coverage area for an MD-IS can be adjacent or adjoin zero or more MD-ISs.</td>
</tr>
<tr>
<td>controls</td>
<td>relationship</td>
<td>The MD-IS controls one or more MDBSs.</td>
</tr>
<tr>
<td>MDBS</td>
<td>class</td>
<td>Mobile Data Base Station. Manages and accesses the radio interface from the network side. Relays and transmits packets sent from the MD-IS.</td>
</tr>
<tr>
<td>adjoins</td>
<td>relationship</td>
<td>An MDBS can be adjacent or adjoin zero or more MDBSs.</td>
</tr>
</tbody>
</table>

Figure 6.4 on page 89 depicts the relationship and structure of the MD-IS, its Serving and Home Functions and their associated information bases, and the associated messaging protocols used by the entities. Each MD-IS contains zero or more MSF’s, one
per NEI currently being served. Correspondingly, each MD-IS contains zero or more MHFs, one per NEI currently being served. The MD-IS contains one registration, location, subscriber, and home domain directory, each of which contain zero or more entries. The MD-IS contains a layer 3 protocol entity used for communicating with IS’s and other MD-ISs as well as a the previously shown MDLP protocol entity used in communication with M-ESs. Finally, the various signalling messages used by the MD-IS are represented by classes as well.
FIGURE 6.4 CDPD Object Model: MD-IS
Table 6.4 and Table 6.5 on page 91 depict the data dictionary for entities in the “CDPD Object Model: MD-IS” diagram.

**TABLE 6.4 MD-IS Data Dictionary**

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD-IS</td>
<td>class</td>
<td>Mobile Data Intermediate System. The CDPD network element and performs routing functions based on knowledge of the current location of the M-ES.</td>
</tr>
<tr>
<td>uses</td>
<td>relationship</td>
<td>The MD-IS uses MNRP Messages for communication with the M-ES.</td>
</tr>
<tr>
<td>L3 Protocol</td>
<td>class</td>
<td>Layer 3 or Network layer protocol used by the MD-IS to exchange NPDUs with adjacent MD-ISs and ISs.</td>
</tr>
<tr>
<td>MNRP Message</td>
<td>class</td>
<td>Mobile Network Registration Protocol Messages.</td>
</tr>
<tr>
<td>uses</td>
<td>relationship</td>
<td>The MD-IS uses MNLP Messages for communication other MD-ISs.</td>
</tr>
<tr>
<td>MNLP Message</td>
<td>class</td>
<td>Mobile Network Location Protocol Messages.</td>
</tr>
<tr>
<td>MSF</td>
<td>class</td>
<td>Mobile Serving Function.</td>
</tr>
<tr>
<td>MHF</td>
<td>class</td>
<td>Mobile Home Function.</td>
</tr>
<tr>
<td>Home Domain Directory</td>
<td>class</td>
<td>An information base that allows a serving MD-IS to determine an M-ESs Location Update Service Address.</td>
</tr>
<tr>
<td>Home Dir Entry</td>
<td>class</td>
<td>An entry in the Home Domain Directory.</td>
</tr>
<tr>
<td>Area Address</td>
<td>attribute</td>
<td>Matching address.</td>
</tr>
<tr>
<td>Address Mask</td>
<td>attribute</td>
<td>Bit mask.</td>
</tr>
<tr>
<td>LUS</td>
<td>attribute</td>
<td>Location Update Service NEI.</td>
</tr>
<tr>
<td>Location Directory</td>
<td>class</td>
<td>An information base specifying the current Forwarding Address of an M-ES.</td>
</tr>
<tr>
<td>Loc Dir Entry</td>
<td>class</td>
<td>An entry in the Location Directory.</td>
</tr>
<tr>
<td>M-ES NEI</td>
<td>attribute</td>
<td>Network Entity Identifier for the M-ES.</td>
</tr>
<tr>
<td>Name</td>
<td>Kind</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RSC</td>
<td>attribute</td>
<td>Registration Sequence Counter.</td>
</tr>
<tr>
<td>FS</td>
<td>attribute</td>
<td>Forwarding service NEI.</td>
</tr>
<tr>
<td>Registration Directory</td>
<td>class</td>
<td>An information base containing an M-ESs Registration Sequence Count and SNPA.</td>
</tr>
<tr>
<td>Reg Dir Entry</td>
<td>class</td>
<td>An entry in the Registration Directory.</td>
</tr>
<tr>
<td>M-ES SNPA</td>
<td>attribute</td>
<td>M-ES TEI.</td>
</tr>
<tr>
<td>Subscriber Directory</td>
<td>class</td>
<td>An information base containing the profile for all CDPD subscribers belonging to a Home MD-IS.</td>
</tr>
<tr>
<td>Sub Dir Entry</td>
<td>class</td>
<td>An entry in the Subscriber Directory.</td>
</tr>
<tr>
<td>AuthInfo</td>
<td>attribute</td>
<td>Valid sequence numbers and random numbers.</td>
</tr>
<tr>
<td>LocationInfo</td>
<td>attribute</td>
<td>Valid service group identifiers for M-ES.</td>
</tr>
<tr>
<td>MDLP</td>
<td>class</td>
<td>Mobile Data Link Protocol. The link layer protocol used in CDPD.</td>
</tr>
<tr>
<td>uses</td>
<td>relationship</td>
<td>MDLP uses MDLP messages to establish data links between an MD-IS and M-ESs.</td>
</tr>
<tr>
<td>MDLP Message</td>
<td>class</td>
<td>Mobile Data Link Protocol Messages.</td>
</tr>
</tbody>
</table>

Figure 6.5 on page 92 through Figure 6.7 on page 93 show the generalization relationships present in the CDPD system. The generalization and subsequent inheritance relationships do not play a part in the concurrency of the system and are not elaborated on further.
FIGURE 6.5 CDPD Object Model: Information Base Generalization

FIGURE 6.6 CDPD Object Model: MNRP Message Generalization
TABLE 6.6 MNRP Message Generalization Data Dictionary

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESH</td>
<td>class</td>
<td>End System Hello.</td>
</tr>
<tr>
<td>ESB</td>
<td>class</td>
<td>End System Bye.</td>
</tr>
<tr>
<td>ESQ</td>
<td>class</td>
<td>End System Query.</td>
</tr>
<tr>
<td>ISC</td>
<td>class</td>
<td>MD-IS Hello Conform.</td>
</tr>
</tbody>
</table>

FIGURE 6.7 CDPD Object Model: MNLP Message Generalization
TABLE 6.7 MNLP Message Generalization Data Dictionary

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDR</td>
<td>class</td>
<td>Redirect Request.</td>
</tr>
<tr>
<td>RDC</td>
<td>class</td>
<td>Redirect Confirm.</td>
</tr>
<tr>
<td>RDQ</td>
<td>class</td>
<td>Redirect Query.</td>
</tr>
<tr>
<td>RDF</td>
<td>class</td>
<td>Redirect Flush.</td>
</tr>
<tr>
<td>RDE</td>
<td>class</td>
<td>Redirect Expiry.</td>
</tr>
</tbody>
</table>

6.1.1.1 System Object Model

The final phase of the object model is the identification of the system object model. The system object model identifies the classes and relationships of the system as opposed to those of its environment. While an argument could be made for the inclusion of the entire CDPD Object Model in a true CDPD system, this case study limits itself to the functions of the MD-IS, excluding that of the layer 3 network protocol entity and MDLP. This focuses the case study on the mobility management functions of the MSF and MHF. Figure 6.8 on page 95 shows the system object model used by this case study. Table 6.8 on page 96 shows the agents data dictionary for the CDPD system. Table 6.9 on page 97 shows the data dictionary for the system interface.
FIGURE 6.8 CDPD Object Model: System Interface
<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MHF</td>
<td>agent</td>
<td>Mobile Home Function of an MD-IS. Note that due to the distributed nature of the CDPD system, the MHF is both a class inside the system and an agent outside of it.</td>
</tr>
<tr>
<td>MSF</td>
<td>agent</td>
<td>Mobile Serving Function of an MD-IS. Note that due to the distributed nature of the CDPD system, the MF is both a class inside the system and an agent outside of it.</td>
</tr>
<tr>
<td>IS</td>
<td>agent</td>
<td>Intermediate System.</td>
</tr>
<tr>
<td>M-ES</td>
<td>agent</td>
<td>Mobile End System.</td>
</tr>
<tr>
<td>L3 Protocol</td>
<td>agent</td>
<td>Layer 3 or Network Layer protocol.</td>
</tr>
<tr>
<td>MDLP</td>
<td>agent</td>
<td>Mobile Data Link Protocol.</td>
</tr>
<tr>
<td>Timer</td>
<td>agent</td>
<td>System Timer.</td>
</tr>
</tbody>
</table>
TABLE 6.9 System Interface Data Dictionary

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>Agent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>esh</td>
<td>operation</td>
<td>M-ES</td>
<td>End System Hello.</td>
</tr>
<tr>
<td>rdr</td>
<td>operation/ event</td>
<td>MHF/ MSF</td>
<td>Redirect Request.</td>
</tr>
<tr>
<td>rdc</td>
<td>operation/ event</td>
<td>MSF/ MHF</td>
<td>Redirect Confirm.</td>
</tr>
<tr>
<td>isc</td>
<td>event</td>
<td>M-ES</td>
<td>MD-IS Hello Confirm.</td>
</tr>
<tr>
<td>rdf</td>
<td>operation/ event</td>
<td>MSF/ MHF</td>
<td>Redirect Flush.</td>
</tr>
<tr>
<td>esb</td>
<td>operation</td>
<td>M-ES</td>
<td>End System Bye.</td>
</tr>
<tr>
<td>rde</td>
<td>operation/ event</td>
<td>MHF/ MSF</td>
<td>Redirect Expiry.</td>
</tr>
<tr>
<td>mhfNpdu</td>
<td>operation</td>
<td>MHF</td>
<td>NPDU to mhf.</td>
</tr>
<tr>
<td>msfNpdu</td>
<td>operation/ event</td>
<td>MSF/ MHF</td>
<td>NPDU to msf.</td>
</tr>
<tr>
<td>mdlpNpdu</td>
<td>event</td>
<td>MDLP</td>
<td>NPDU to MDLP.</td>
</tr>
</tbody>
</table>

6.1.2 Interface Model

The Fusion interface model attempts to capture the behavior present via the life-cycle and operation models. The system is thought of as an active entity which interacts with other active entities called agents. The agents in the system are defined as those classes in the object model which are outside of the system boundary, otherwise known as the environment with which the system interacts. For CDPD, the agents of interest include the M-ES, other MD-ISs, and ISs. The system object model and the scenario diagrams defined in Chapter 5 help identify the events or communications between the system and its environment. In the Fusion analysis phase, all events are considered asynchronous as the sender does not have to wait for the event to be received. Typically, these events cause a change of system state and the output of events. An input event and its corresponding state changes and output events are collectively referred to as a system operation. Fusion imposes the restriction that only one system operation may
be in effect at a time and for the analysis phase this is acceptable. Finally, the set of system operations and output events a system can receive and produce are collectively known as the system’s interface, hence the name interface model.

6.1.2.1 Life-Cycle Model

The life-cycle model shows the allowable sequencing of system operations and events. Input events which arrive outside of this sequence are ignored by the system. The life-cycle model in Figure 6.9 characterizes the system operations of the MD-IS class.

Note that the life-cycle model shown below has been simplified by focusing only on the mobility management aspects of the MSF and MHF at a network layer perspective. Operations for the initialization of the system and its components, interactions with the layer 2 entities, and error reporting activities to the layer 3 network entity have all been intentionally omitted to reduce the size of the problem domain.

![Life Cycle Model Diagram]

FIGURE 6.9 Life Cycle Model

In summary, the MD-IS’s life-cycle is formed of interleaved MSF and MHF operations. The MSF’s life-cycle is formed by interleaved Report Location, Flush
Registration, and Readdress operations. A Report Location operation consists of either an End System Hello entering the system from an M-ES followed by output events to the system timer to restart the holding timer and a Redirect Request to an MD-IS, or a Redirect Confirm entering the system from an MD-IS followed by an output event of an MD-IS Hello Confirm to an M-ES.

The Flush Registration operation can be initiated by either an incoming Redirect Flush event from an MD-IS, an End System Bye from an M-ES, or the expiry of the holding timer. In either of the last two cases, a Redirect Expiry event is send to an MD-IS.

The Readdress operation is always triggered by the arrival of an NPDU on the MD-IS Forwarding Service NEI. The NPDU is either transmitted to an M-ES in the form of a MDLP NPDU or rejected by sending an Redirect Expiry event to the MD-IS.

The MHF’s life cycle is formed by interleaved Record Location, Flush Location, and Redirection operations. The Record Location operation is one of the most complex operations in the CDPD system. This operation always begins with the arrival of an Redirect Request from an MD-IS. This Redirect Request may either be forwarded to another MD-IS, rejected by sending a Redirect Flush event to the MD-IS, or accepted resulting in a Restart event to the system timer, a Redirect Confirm to the MD-IS, and a possible Redirect Flush to the previous MD-IS.

The Flush Location operation begins with the arrival of a Redirect Expiry or expiration of the holding timer. The Flush Location operation produces no output events.

The Redirection operation is invoked by the arrival of an NPDU at the MD-IS for an M-ES currently served elsewhere. The NPDU is encapsulated and sent out to the MD-IS.

6.1.2.2 Operational Model

The operational model provides additional details and conditional logic in characterizing the effect of each system operation on the state of the system and the output events it causes. Table 6.10 on page 101 through Table 6.19 on page 108 provide the schemas for each operation identified in Section 6.1.2.1 on page 98. Again note that the
operational model builds upon the life cycle previously presented and does not attempt to model the entire CDPD system.
TABLE 6.10 esh Schema

<table>
<thead>
<tr>
<th>Operation</th>
<th>esh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>MNRP End System Hello. This event is sent from an M-ES when (1) the data link is initially established (i.e., initial TEI assignment), (2) Expiration of the M-ES configuration timer, (3) Expiration of the M-ES response timer, (4) Subsequent activation of additional NEIs, and (5) Upon receipt of an ESQ for an NEI currently active on the M-ESs data link (i.e., TEI).</td>
</tr>
</tbody>
</table>
| Reads | • supplied esh.m-esNei  
      • supplied esh.spna  
      • supplied esh.rsc  
      • supplied esh.authInfo  
      • homeDomainDirectory  
      • cell.locationInfo  
      • md-is.fs  
      • md-is.lus |
| Changes | • new registrationDirectoryEntry  
          • new rdr |
| Sends | • Timer: {msfHoldingTimerRestart}  
      • MHF: {rdr} |
| Assumes | • A new registration_directory_entry is created if the supplied {esh.source_network_address, esh.spna} is not already present  
          • The holding timer for the {esh.m-esNei, esh.spna} pair is re-started.  
          • The registrationDirectoryEntry.rsc is updated with the esh.rsc.  
          • A new rdr is created.  
          • The rdr.m-esNei is set to the esh.m-esNei, the rdr.rsc is set to the esh.rsc, the rdr.fs is set to the md-is.fs, the esh.authInfo and cell.locationInfo are added to the rdr.  
          • The rdr.destinationAddress is set to the lus address determined by looking up the esh.m-esNei in the homeDomainDirectory. The rdr.sourceAddress is set to the md-is.lus. |
<table>
<thead>
<tr>
<th>Operation</th>
<th>rdc</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>MNLP Redirect Confirm. This event is sent from the home MD-IS to acknowledge an M-ES’s presence at its current location and to notify the serving MD-IS of the home MD-IS’s willingness or ability to provide network forwarding service at its current location.</td>
</tr>
<tr>
<td><strong>Reads</strong></td>
<td>• supplied rdc.m-esNei • supplied rdc.authUpdate • supplied rdc.resultCode • supplied rdc.configurationTimer • registrationDirectoryEntry</td>
</tr>
<tr>
<td><strong>Changes</strong></td>
<td>• new isc • registrationDirectoryEntry</td>
</tr>
<tr>
<td><strong>Sends</strong></td>
<td>• M-ES: {isc}</td>
</tr>
<tr>
<td><strong>Assumes</strong></td>
<td>• A registrationDirectoryEntry for rdc.m-esNei is present. • rdc.rsc = rdc.m-esNei’s registrationDirectoryEntry.rsc</td>
</tr>
<tr>
<td><strong>Result</strong></td>
<td>• A new isc is created. • The isc.m-esNei is set to the rdc.m-esNei. • The isc.authUpdate is set to the rdc.authUpdate, if present. • The isc.resultCode is set to the rdc.resultCode, if present. • The isc.configurationTimer is set to the rdc.configurationTimer, if present. • The isc is routed to the SNPA address stored in the corresponding registrationDirectoryEntry. • If the rdc.resultCode indicates service being denied by the home MD-IS, then the corresponding registrationDirectoryEntry is removed.</td>
</tr>
</tbody>
</table>
TABLE 6.12 rdr Schema

<table>
<thead>
<tr>
<th>Operation</th>
<th>rdr</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>MNLP Redirect Request. This event is sent by a serving MD-IS to register the existence and reachability of a network address of an M-ES with the home MD-IS.</td>
</tr>
</tbody>
</table>
| **Reads** | • supplied rdr.m-esNei, supplied rdr.rsc, supplied rdr.authInfo, supplied rdr.locationInfo, supplied rdr.fs  
  • subscriberDirectory, homeDomainDirectory, locationDirectoryEntry  
  • md-is.lus |
| **Changes** | • new rdf, new rdc  
  • new locationDirectoryEntry, locationDirectoryEntry.fs, locationDirectoryEntry.rsc |
| **Sends** | • MSF: {rdc, rdf}  
  • MHF: {rdr}  
  • Timer: {mhfHoldingTimerRestart} |
| **Result** | • If the rdr.m-esNei is not present in the subscriberDirectory, then the rdr is re-routed to the address determined by looking up the rdr.m-esNei in the homeDomainDirectory.  
  • Otherwise, if the rdr.rsc indicates an out-of-sequence registration attempt when compared with the locationDirectoryEntry.rsc, then an rdf is constructed with rdf.destinationAddress set to rdr.sourceAddress and rdf.sourceAddress set to md-is.lus.  
  • Otherwise, the rdr.authInfo and rdr.locationInfo are authorized via the subscriberDirectoryEntry corresponding to rdr.m-esNei. A new resultCode and authUpdate are created.  
  • A new locationDirectoryEntry is created if none previously exists. The locationDirectoryEntry.rsc is set to the rdr.rsc and the locationDirectoryEntry.fs is set to the rdr.fs.  
  • The mhfHoldingTimer is restarted.  
  • A new rdc is constructed with the rdc.rsc set to the rdr.rsc, the homeInfo set to subscriberDirectoryEntry.homeInfo, the rdc.authUpdate set to the new authUpdate, and the rdc.resultCode is set to the new resultCode.  
  • The rdc is routed with the rdc.destinationAddress set to the rdr.sourceAddress and the rdc.sourceAddress set to md-is.lus. |
TABLE 6.13 rdf Schema

<table>
<thead>
<tr>
<th>Operation</th>
<th>rdf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>MNLP Redirect Flush. This event is sent by a home MD-IS to notify a serving MD-IS that a Network Address previously forwarded to that location has moved and shall no longer be forwarded there.</td>
</tr>
</tbody>
</table>
| Reads | • supplied rdf.rsc  
        • registrationDirectoryEntry.rsc |
| Changes | • registrationDirectory |
| Sends | |
| Assumes | • A registrationDirectoryEntry for rdf.m-esNei is present.  
          • rdf.rsc = rdf.m-esNei’s registrationDirectoryEntry.rsc |
<p>| Result | • The corresponding registrationDirectoryEntry is removed. |</p>
<table>
<thead>
<tr>
<th>Operation</th>
<th>esb</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>MNRP End System Bye. This event is sent by an M-ES to deregister the existence and reachability of an NEI.</td>
</tr>
</tbody>
</table>
| **Reads** | • supplied esb.m-esNei  
• registrationDirectoryEntry.rsc  
• homeDomainDirectory  
• md-is.lus  
| **Changes** | • new rde  
• registrationDirectoryEntry  
| **Sends** | • MHF: \{rde\}  
| **Assumes** | • A registrationDirectoryEntry for esb.m-esNei is present.  
| **Result** | • A new rde is created.  
• The rde.m-esNei is set to the esb.m-esNei, the rde.rsc is set to the corresponding registrationDirectoryEntry.rsc.  
• The rde.destinationAddress is set to the lus address determined by looking up the esb.m-esNei in the homeDomainDirectory. The rde.sourceAddress is set to the md-is.lus.  
• The corresponding registrationDirectoryEntry is removed. |
<table>
<thead>
<tr>
<th>Operation</th>
<th>msfHoldingTimerExpiry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Expiry of the registrationDirectoryEntry holding timer.</td>
</tr>
</tbody>
</table>
| Reads | • supplied msfHoldingTimerExpiry.m-esNei  
• registrationDirectoryEntry.rsc  
• homeDomainDirectory  
• md-is.lus |
| Changes | • new rde  
• registrationDirectoryEntry |
| Sends | • MHF: {rde} |
| Assumes | • A registrationDirectoryEntry for msfHoldingTimerExpiry.m-esNei is present. |
| Result | • A new rde is created.  
• The rde.m-esNei is set to the corresponding registrationDirectory.m-esNei, the rde.rsc is set to the corresponding registrationDirectoryEntry.rsc.  
• The rde.destinationAddress is set to the lus address determined by looking up the corresponding registrationDirectory.m-esNei in the homeDomainDirectory. The rde.sourceAddress is set to the md-is.lus.  
• The corresponding registrationDirectoryEntry is removed. |
### TABLE 6.16 mhfHoldingTimerExpiry Schema

<table>
<thead>
<tr>
<th>Operation</th>
<th>mhf_holding_timer_expiry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Expiry of the location_directory_entry holding timer.</td>
</tr>
<tr>
<td>Reads</td>
<td>• supplied mhfHoldingTimerExpiry.m-esNei</td>
</tr>
<tr>
<td>Changes</td>
<td>• locationDirectoryEntry</td>
</tr>
<tr>
<td>Sends</td>
<td></td>
</tr>
<tr>
<td>Assumes</td>
<td>• A locationDirectoryEntry for mhfHoldingTimerExpiry.m-esNei is present.</td>
</tr>
<tr>
<td>Result</td>
<td>• The corresponding locationDirectoryEntry is removed.</td>
</tr>
</tbody>
</table>

### TABLE 6.17 rde Schema

<table>
<thead>
<tr>
<th>Operation</th>
<th>rde</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>MNLP Redirect Expiry. This event is sent by a serving MD-IS to notify a home MD-IS that a Network Address previously forwarded to that location has moved or deregistered and should no longer be forwarded there.</td>
</tr>
</tbody>
</table>
| Reads     | • supplied rde.m-esNei  
• supplied rde.rsc |
| Changes   | • locationDirectoryEntry |
| Sends     | • |
| Assumes   | • A locationDirectoryEntry for rde.m-esNei is present.
• rde.rsc = corresponding locationDirectoryEntry.rsc |
| Result    | • The corresponding locationDirectoryEntry is removed. |
### TABLE 6.18 mhfNpdu Schema

<table>
<thead>
<tr>
<th>Operation</th>
<th>mhf_npdu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>NPDUs arriving at an M-ES’s home address for M-ESs currently served elsewhere are readdressed and forwarded to their serving MD-IS.</td>
</tr>
</tbody>
</table>
| Reads | • supplied mhfNpdu.m-esNei  
  • locationDirectory  
  • md-is.fs |
| Changes | • new msfNpdu |
| Sends | • MSF: {msfNpdu} |
| Assumes | • mhfNpdu.m-esNei is not present in the registrationDirectory  
  • mhfNpdu.m-esNei is present in the locationDirectory |
| Result | • Encapsulate the mhfNpdu in a new msfNpdu with its destinationAddress set to the corresponding locationDirectoryEntry.fs and the sourceAddress set to md-is.fs. |

### TABLE 6.19 msfNpdu Schema

<table>
<thead>
<tr>
<th>Operation</th>
<th>msfNpdu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>NPDUs arriving on the MSF’s forwarding service address are decapsulated and re-routed to the M-ES’s SNPA.</td>
</tr>
</tbody>
</table>
| Reads | • registrationDirectoryEntry  
  • supplied msfNpdu.m-esNei |
| Changes | • new rde |
| Sends | MHF: {rde}  
  MDLP: {mdlpNpdu} |
| Assumes | |
| Result | • If a registrationDirectoryEntry exists for the decapsulated msfNpdu.m-esNei, then the NPDUs is routed to registrationDirectoryEntry.snpa. |
  • Otherwise, an rde is constructed and routed to the sending MHF. |
6.2 Design

The Fusion design process takes the model built during analysis and designs a system with the described behavior. The object interaction graphs show how the objects cooperate to provide the described system level functions. The visibility graphs show how the objects reference each other. The class descriptions document each class. Finally, the inheritance graphs show how the classes are related through inheritance.

6.2.1 Object Interaction Graphs

Each system operation identified in the analysis phase has an associated object interaction graph. The object interaction graphs show what objects are involved in the operation and how they collaborate. The first step is to identify the objects and/or agents involved in a system operation. Next, the role of each object is determined. The controller role is assigned to the object receiving the system operation message. All other objects are assigned the role of collaborator. Finally, the functionality of the operation is distributed appropriately across the various object involved and recorded in the object interaction graph. The object interaction graphs are broken into two sections: one for events which are routed to the Mobile Serving Function and one for events which are routed to the Mobile Home Function.

Several of the concurrency extensions from Chapter 4 are used in the object interaction graphs which follow. As per the rules outlined in Chapter 4, the objects which receive asynchronous events from the environment are noted as ‘Active’. Further more, objects which receive asynchronous internal events from active object are also noted as active. These rules require the MD-IS, MSF, and MHF all be marked as active objects containing their own thread of control.

In addition, several synchronous messaging situations which are identified via analysis of method semantics in Section 6.2.3 on page 124 are also denoted on the object interaction graph. More information regarding the identification process is given in Section 6.2.3.
6.2.1.1 MSF Object Interaction Graphs

All system operations directed toward the MSF actually begin as an asynchronous event arriving at the MD-IS in the form of a network protocol data unit (NPDU). The physical NPDU is actually delivered to the network layer protocol entity first (not shown) which then signals the MD-IS of its arrival. The MD-IS decodes this NPDU as an MSF event and routes it to the appropriate MSF object currently serving the M-ES NEI associated with the NPDU, creating a new MSF object in the process if required.

![Figure 6.10 Object Interaction Graph: MSF NPDU](image)

Figure 6.10 shows the object interaction graph for this operation. Note that event "(3) msfEvent(parms)" is a generic event which will be replaced in Figure 6.11 on page 111 through Figure 6.16 on page 116 with the actual decoded MSF event and associated parameters. Note also that the message sequence numbering in Figure 6.11 through Figure 6.16 starts at ‘(3.1)’ as a direct result of Figure 6.10.
Figure 6.11 shows the object interaction graph for the End System Hello (ESH) operation. The MD-IS has previously decoded an incoming NPDU as an ESH and selected the appropriate MSF object to handle the asynchronous event. The MSF object begins by querying (3.1) the registrationDirectory for the presence of the M-ES NEI. If not found then the exist message returns a ‘nil’ regDirID indicating that a entry should be created (3.2). The entry (existing or new) is updated with the ESH’s RSC value (3.3), causing the entry’s associated holding timer to be restarted (3.3.1). The cell object is then queried (3.4) to determine the M-ES’s physical location and a Redirect Request (RDR) is constructed (3.5) containing the M-ES NEI, RSC, authentication credentials, and location information as well as the MD-IS’s Forwarding Service (FS) NEI. The destination for the RDR is retrieved (3.6) from the homeDomainDirectory and the new NPDU is transmitted (3.7) asynchronously onto the network (3.7.1) through the network layer protocol entity and attached IS’s (not shown).
Figure 6.12 shows the object interaction graph for the End System Bye (ESB) operation. The MD-IS has previously decoded an incoming NPDU as an ESB and selected the appropriate MSF object to handle the asynchronous event. The MSF object begins by querying (3.1) the registrationDirectory for the presence of the M-ES NEI. If not found then the event is discarded, otherwise the RSC is retrieved (3.2) and a Redirect Expiry (RDE) is created (3.3) containing the M-ES's NEI and the RSC value. The destination for the RDE is retrieved (3.4) from the homeDomainDirectory and the new NPDU is transmitted (3.5) asynchronously onto the network (3.5.1) through the network layer protocol entity and attached IS's (not shown). Finally, the registrationDirectoryEntry is removed (3.6) from the registrationDirectory.
Figure 6.13 shows the object interaction graph for the Redirect Confirm (RDC) operation. The MD-IS has previously decoded an incoming NPDU as an RDC and selected the appropriate MSF object to handle the asynchronous event. The MSF object begins by querying (3.1) the registrationDirectory for the presence of the M-ES NEI. If not found then the event is discarded, otherwise the registrationDirectoryEntry is queried (3.2) for its stored RSC value. This value is compared against the one contained in the RDC. If they do not match, the RDC is discarded, otherwise, a new ISC is created (3.3) containing the M-ES's NEI and the authentication update, result code, and M-ES configuration timer value contained in the RDC. The registrationDirectoryEntry is queried again (3.4) for the SNPA of the M-ES and the ISC is transmitted (3.5) to the MDLP entity and onto the M-ES (3.5.1) asynchronously through the MDBS (not shown). Finally, if the Result Code contained in the RDC indicates other than a successful registration, then the registrationDirectoryEntry is removed (3.6).
Figure 6.14 shows the object interaction graph for the Redirect Flush (RDF) operation. The MD-IS has previously decoded an incoming NPDU as an RDF and selected the appropriate MSF object to handle the asynchronous event. The MSF object begins by querying (3.1) the registrationDirectory for the presence of the M-ES NEI. If not found then the event is discarded, otherwise the registrationDirectoryEntry is queried (3.2) for its stored RSC value. This value is compared against the one contained in the RDF. If they do not match, the RDF is discarded, otherwise the registrationDirectoryEntry is removed (3.3).
Figure 6.15 shows the object interaction graph for the msfHoldingTimer expiry operation. The MD-IS has previously received an incoming event from the system timer and selected the appropriate MSF object to handle the asynchronous event. The MSF object begins by querying (3.1) the registrationDirectory for the presence of the M-ES NEI. If not found then the event is discarded, otherwise the RSC is retrieved (3.2) and a Redirect Expiry (RDE) is created (3.3) containing the M-ES’s NEI and the RSC value. The destination for the RDE is retrieved (3.4) from the homeDomainDirectory and the new NPDU is transmitted (3.5) onto the network (3.5.1) asynchronously through the network layer protocol entity and attached IS’s (not shown). Finally, the registrationDirectoryEntry is removed (3.6) from the registrationDirectory.
Figure 6.16 shows the object interaction graph for the msfNPDU operation. The MD-IS has previously decoded an incoming NPDU as a msfNPDU and selected the appropriate MSF object to handle the asynchronous event. The MSF object begins by querying (3.1) the registrationDirectory for the presence of the M-ES NEI. If the entry is not found then the NPDU is discarded. Otherwise, a new mdlpNPDU is created (3.2) containing the data from the msfNPDU. The registrationDirectoryEntry is queried (3.3) for the SNPA of the M-ES and the mdlpNPDU is transmitted (3.4) to the MDLP entity and onto the M-ES (3.4.1) asynchronously through the MDBS (not shown).

6.2.1.2 MHF Object Interaction Graphs

All system operations directed toward the MHF actually begin as an asynchronous event arriving at the MD-IS in the form of a network protocol data unit (NPDU). The physical NPDU is actually delivered to the network layer protocol entity first (not shown) which then signals the MD-IS of its arrival. The MD-IS decodes this NPDU as an MHF
event and routes it to the appropriate MHF object currently serving the M-ES NEI associated with the NPDU, creating a new MHF object in the process if required.

FIGURE 6.17 Object Interaction Graph: MHF NPDU

Figure 6.17 shows the object interaction graph for this operation. Note that event "(3) mhfEvent(parms)" is a generic event which will be replaced in Figure 6.18 on page 117 through Figure 6.21 on page 120 with the actual decoded mhf event and associated parameters. Note also that the message sequence numbering in Figure 6.18 through Figure 6.21 starts at ‘(3.1)’ as a direct result of Figure 6.17.

FIGURE 6.18 Object Interaction Graph: RDR to wrong MD-IS
The Redirect Request (RDR) operation is shown in Figure 6.18 through Figure 6.20. The actual RDR operation should be taken as the union of these figures. Figure 6.18 on page 117 shows the object interaction graph for the Redirect Request (RDR) operation arriving at the wrong MD-IS. The MD-IS has previously decoded an incoming NPDU as an RDR and selected the appropriate MHF object to handle the asynchronous event. The MHF object begins by querying (3.1) the subscriberDirectory for the presence of the M-ES NEI. If not found then a new Redirect Request (RDR) is constructed (3.2) containing an exact copy of all the parameters present in the original RDR. The destination for the RDR is retrieved (3.3) from the homeDomainDirectory and the new NPDU is transmitted (3.4) onto the network (3.4.1) asynchronously through the network layer protocol entity and attached IS’s (not shown).

Figure 6.19 shows the object interaction graph for the Redirect Request (RDR) which fails RSC checking. The MD-IS has previously decoded an incoming NPDU as an RDR and selected the appropriate MHF object to handle the asynchronous event. The MHF object begins by querying (3.1) the subscriberDirectory for the presence of the

![Object Interaction Graph: RDR fails RSC checking](image-url)
M-ES NEI. Next, the locationDirectory is queried (3.2) for the presence of the M-ES NEI. If not found, the RDR is discarded, otherwise the RSC is retrieved (3.3). Assuming the RSC contained in the RDR and to RSC retrieved from the locationDirectoryEntry do not check, then a Redirect Flush (RDF) is created (3.4) containing the M-ES’s NEI and the RSC from the RDR. The destination for the RDF is determined from the sourceAddress of the RDR and the new NPDU is transmitted (3.5) onto the network (3.5.1) asynchronously through the network layer protocol entity and attached IS’s (not shown).

Figure 6.20 shows the object interaction graph for the Redirect Request (RDR) operation. The MD-IS has previously decoded an incoming NPDU as an RDR and selected the appropriate MHF object to handle the asynchronous event. The MHF object begins by querying (3.1) the subscriberDirectory for the presence of the M-ES NEI. Next, the locationDirectory is queried (3.2) for the presence of the M-ES NEI. If not found then the exist message returns a ‘nil’ locDirID indicating that a entry should be created (3.3). The entry (existing or new) is updated with the RDR’s RSC value (3.4), causing the
entry's associated holding timer to be restarted (3.4.1). Next, the MHF passes the M-ES’s Authentication Credentials and Location Information to the subscriberDirectoryEntry for authorization (3.5) and the subscriberDirectoryEntry’s Home Information is retrieved (3.6). A new Redirect Confirm (RDC) is created (3.7) containing the M-ES’s NEI and RSC, the Home Information retrieved from the subscriberDirectory, and the Authentication Update and Result Code from the authorization process. The destination for the RDC is determined from the sourceAddress of the RDR and the new NPDU is transmitted (3.8) onto the network (3.8.1) asynchronously through the network layer protocol entity and attached IS’s (not shown). Finally, if the Result Code generated by the authorization process indicates other than a successful registration, then the locationDirectoryEntry is removed (3.9).

![Diagram of Object Interaction Graph: RDE](image)

**FIGURE 6.21 Object Interaction Graph: RDE**

Figure 6.21 shows the object interaction graph for the Redirect Expiry (RDE) operation. The MD-IS has previously decoded an incoming NPDU as an RDE and selected the appropriate MHF object to handle the asynchronous event. The MHF object begins by querying (3.1) the locationDirectory for the presence of the M-ES NEI. If not found then the event is discarded, otherwise the locationDirectory is queried (3.2) for its stored RSC value. This value is compared against the one contained in the RDE. If they do not match, the RDE is discarded, otherwise the locationDirectoryEntry is removed (3.3).
FIGURE 6.22 Object Interaction Graph: MHF Holding Timer

Figure 6.22 shows the object interaction graph for the mhfHoldingTimer expiry operation. The MD-IS has previously received an incoming event from the system timer and selected the appropriate MHF object to handle the asynchronous event. The MHF object begins by querying (3.1) the locationDirectory for the presence of the M-ES NEI. If not found then the event is discarded, otherwise the locationDirectoryEntry is removed (3.2) from the locationDirectory.

FIGURE 6.23 Object Interaction Graph: mhfNPDU
Figure 6.23 on page 121 shows the object interaction graph for the mhfNPDU operation. The MD-IS has previously decoded an incoming NPDU as a mhfNPDU and selected the appropriate MHF object to handle the asynchronous event. The MHF object begins by querying (3.1) the locationDirectory for the presence of the M-ES NEI. If the entry is not found then the NPDU is discarded. Otherwise, a new msfNPDU is created (3.2) containing the data from the mhfNPDU. The locationDirectoryEntry is queried (3.3) for the Forwarding Service (FS) NEI of the MD-IS and the msfNPDU is transmitted (3.4) onto the network (3.4.1) asynchronously to the network layer protocol entity and attached IS’s (not shown).

6.2.2 Visibility Graphs

Figure 6.24 on page 123 shows the visibility graph for the CDPD system. Several items are of note in the visibility graph. First, the life times of the vast majority of the classes in the CDPD system are bound to that of the MD-IS class. The existence of the Mobile Serving and Mobile Home Functions as well as the various information bases all depend on the existence of the MD-IS.

Second is the predominance of dynamic references. The only permanent references are found within the scope of the bound life time classes. The third item of note is the shared and exclusive references. The MD-IS references to the MSF and MHF classes are considered exclusive but more important is the MHF’s exclusive reference to the locationDirectoryEntry and subscriberDirectoryEntry and the MSF’s exclusive access to the registrationDirectory. While these classes are contained in a collection, only one of the respective MSF or MHF classes should ever reference them at any one time. This helps define the type of synchronous messaging used in the object interaction graphs. Finally, the collection of MSFs and MHFs should be noted as this proves essential for enabling a vast majority of the concurrency present in the system by allowing one MHF or MSF object to concurrently service each M-ES NEI present in the system at any one time.
FIGURE 6.24 Visibility Graph
6.2.3 Class Descriptions

The final step in this case study is to collect the information represented in the object interaction graphs and visibility graph and construct the initial class descriptions.

```
class Active md-isClass
    attribute md-isId : md-isIdType
    attribute subscriberDirectory : bound subscriberDirectoryClass
    attribute registrationDirectory : bound registrationDirectoryClass
    attribute locationDirectory : bound locationDirectoryClass
    attribute homeDomainDirectory : bound homeDomainDirectoryClass
    attribute mhf : exclusive bound col mhfClass
    attribute msf : exclusive bound col msfClass
    attribute lus : neiType
    attribute fs : neiType
    attribute msfHoldingTimer : timerValueType
    attribute mhfHoldingTimer : timerValueType
    method MaxCon msfExist(nei : neiType) : msfIdType
    method MaxCon mhfExist(nei : neiType) : mhfIdType
    method MaxSeq npdu()
endclass
```

**FIGURE 6.25 Class Description: md-isClass**

Figure 6.25 shows the class description of the MD-IS class. From the object interaction graphs, the MD-IS is denoted as containing its own thread of control by the keyword ‘Active’. The MD-IS class includes a ID attribute for reference purposes, as do all the classes in the CDPD system. The MD-IS class also includes the information bases present in the CDPD system. Note that the life times of these information bases are bound to that of the MD-IS itself. Next note the collection of MHF and MSF classes. While these classes are actually created and destroyed quite frequently as M-ES NEIs enter and leave the CDPD system, their life times are also ultimately bound to that of the MD-IS. Also
present in the MD-IS class are several simple attributes for the LUS and FS NEIs of the
MD-IS and the values for the various holding timers. The MD-IS class contains three
methods. 'msfExist' and 'mhfExist' are used internally by the MD-IS to ascertain the
existence of MSF and MHF objects respectively. Both of these methods are declared as
maximally concurrent, allowing multiple threads to be present at any one time. The
‘npdu’ method allows the MD-IS to accept NPDUs from the network layer protocol and
MDLP entities. This method is declared as maximally sequential, allowing only one
thread of control to be present at a time. Consequently, the npdu method may accept only
synchronous or asynchronous messages.

```
class Active msfClass
    attribute nei : neiType
    attribute md-isId : md-isIdType
    method create(nei : neiType, md-isId : md-isIdType)
    method MaxSeq esh(spna : snpaType, rsc : rscType, authInfo :
        authInfoType)
    method MaxSeq esb(md-isIdType)
    method MaxSeq rdc(authUpdateParameter :
        authUpdateParameterType, resultCode : resultCodeType,
        configTimer : configTimerType)
    method MaxSeq rdf(rsc : rscType)
    method MaxSeq rdq(rsc : rscType)
    method MaxSeq msfHoldingTimer()
    method MaxSeq npdu(npduBuffer : npduBufferType)
    MutEx(esh, esb, rdc, rdf, rdq, msfHoldingTimer, npdu)
endclass
```

**FIGURE 6.26 Class Description: msfClass**

Figure 6.26 shows the class description for the MSF class. From the object
interaction graphs, the MSF is denoted as containing its own thread of control by the
keyword 'Active'. Note that the MSF is essentially stateless containing attributes only for
the NEI it is currently serving and the MD-IS which created it. The use of the maximally sequential and complete mutual exclusion notations make this object completely sequential in nature. As a consequence, the MSF class’s methods may receive only synchronous or asynchronous messages. Concurrency within the MSF Class is derived by allowing multiple instantiations of the class, one per NEI being served. Finally, the MSF class contains the implicit behavior that it ceases to exist when no messages are enqueued for its methods to receive.

```plaintext
class Active mhfClass
    attribute nei : neiType
    attribute md-isId : md-isIdType
    method create(nei : neiType, md-isId : md-isIdType)
    method MaxSeq rdr(rsc : rscType, fs : neiType, authInfo : authInfoType, locationInfo : locationInfoType)
    method MaxSeq rdf(rsc : rscType)
    method MaxSeq mhfHoldingTimer()
    method MaxSeq npdu(npduBuffer : npduBufferType)
    Mutex(rdr, rdf, mhfHoldingTimer, npdu)
endclass
```

**FIGURE 6.27 Class Description: mhfClass**

Figure 6.27 shows the class description for the MHF class. From the object interaction graphs, the MHF is denoted as containing its own thread of control by the keyword 'Active'. Note that the MHF is essentially stateless containing attributes only for the NEI it is currently serving and the MD-IS which created it. The use of the maximally sequential and complete mutual exclusion notations make this object completely sequential in nature. As a consequence, the MHF class’s methods may receive only synchronous or asynchronous messages. Concurrency within the MHF Class is derived by allowing multiple instantiations of the class, one per NEI being served. Finally, the MHF
class contains the implicit behavior that it ceases to exist when no messages are enqueued for its methods to receive.

```plaintext
class Passive subscriberDirectoryClass
    attribute subscriberDirectoryId : subscriberDirectoryIdType
    attribute subscriberDirectoryEntry : bound col
        subscriberDirectoryEntryClass
    method MaxCon exist(nei : networkEntityIdentifierType) :
        subscriberDirectoryEntryIdType
endclass
```

**FIGURE 6.28 Class Description: subscriberDirectoryClass**

Figure 6.28 shows the class description for the subscriber directory. The subscriber directory class, much like all the information bases in the CDPD system, consists of a collection of directory entries, which are ultimately bound to the life time of the subscriber directory, and one method for ascertaining the existence of particular subscribers. Note that the ‘exist’ method is denoted as maximally concurrent, allowing multiple threads of control to be present at any one time.
class **Passive** subscriberDirectoryEntryClass **MaxCon**

- attribute subscriberDirectoryEntryClassId : subscriberDirectoryEntryClassIdType
- attribute nei : networkEntityIdentifierType
- attribute homeInfo : homeInfoType
- attribute authInfo : authInfoType
- attribute validLocations : col locationInfoType
- attribute configTimer : configTimerType

method **MaxCon** getHomeInfo() : homeInfoType

method **MaxSeq** authorize(authInfo : authInfoType, locationInfo : locationInfoType, authUpdate : authUpdateType) : resultCodeType

endclass

**FIGURE 6.29 Class Description: subscriberDirectoryEntryClass**

Figure 6.29 shows the class description for the subscriber directory entry class. The subscriber directory entry contains information on a per NEI basis. Its attributes include the M-ES’s NEI, home information, authentication information, configuration timer value, and a collection of valid locations. The subscriber directory entry contains two methods, one maximally concurrent method for retrieving an M-ES’s home information, and one maximally sequential method for authorizing an M-ES’s service. Consequently, messages to the ‘authorize’ method must be either synchronous or asynchronous.
class Passive homeDomainDirectoryClass
    attribute homeDomainDirectoryId : homeDomainDirectoryIdType
    attribute homeDomainDirectoryEntry : bound col
        homeDomainDirectoryEntryClass
    method MaxCon getLus(nei : neiType) : neiType
endclass

FIGURE 6.30 Class Description: homeDomainDirectoryClass

Figure 6.30 shows the class description for the home domain directory class. The home domain directory class, much like all the information bases in the CDPD system, consists of a collection of directory entries, which are ultimately bound to the lifetime of the home domain directory. The home domain directory contains one method for ascertaining the destination location update service NEI for a given M-ES NEI. Note that the ‘getLus’ method is denoted as maximally concurrent, allowing multiple threads of control to be present at any one time.

class Passive homeDomainDirectoryEntryClass
    attribute areaAddress : areaAddressType
    attribute areaMask : areaMaskType
    attribute lus : neiType
endclass

FIGURE 6.31 Class Description: homeDomainDirectoryEntryClass

Figure 6.31 shows the class description for the home domain directory entry class. The home domain directory entry contains information used to determine an M-ES NEI’s LUS NEI. The algorithm detailing the use of the area address and area mask for determining an LUS are beyond the scope of this study.
class **Passive** registrationDirectoryClass

attribute registrationDirectoryId : registrationDirectoryIdType

attribute registrationDirectoryEntry : bound col

registrationDirectoryEntryClass

method **MaxCon** exist(nei : neiType):

registrationDirectoryEntryIdType

diclass

FIGURE 6.32 Class Description: registrationDirectoryClass

Figure 6.32 shows the class description for the registration directory. The registration directory class, much like all the information bases in the CDPD system, consists of a collection of directory entries, which are ultimately bound to the life time of the registration directory, and one method for ascertaining the existence of particular registration directory entry. Note that the ‘exist’ method is denoted as maximally concurrent, allowing multiple threads of control to be present at any one time.
class **Passive** registrationDirectoryEntryClass

attribute registrationDirectoryEntryId : registrationDirectoryEntryIdType

attribute nei : neiType

attribute rsc : rscType

attribute snpa : snpaType

method create(nei : neiType, snpa : snpaType)

method remove()

method **MaxSeq** updateRsc(rsc : rscType)

method **MaxCon** getRsc() : rscType

method **MaxCon** getSnpa() : snpaType

**MutEx** (updateRsc, getRsc)

endclass

**FIGURE 6.33 Class Description: registrationDirectoryEntryClass**

Figure 6.33 shows the class description for the registration directory entry class. The registration directory entry contains information on a per NEI basis. Its attributes include the M-ES’s NEI, RSC, and SNPA. The registration directory entry contains three methods in addition to its ‘create’ and ‘remove’ methods, two maximally concurrent methods for retrieving an M-ES’s RSC and SNPA, and one maximally sequential method for updating an M-ES’s RSC. Note that the methods for accessing and updating the RSC are mutually exclusive. Consequently, messages to the ‘updateRsc’ and ‘getRsc’ methods must be either synchronous or asynchronous.
class Passive locationDirectoryClass

    attribute locationDirectoryId : locationDirectoryIdType
    attribute locationDirectoryEntry : bound col
        locationDirectoryEntryClass
    method MaxCon exist(nei : neiType) :
        locationDirectoryEntryIdType

endclass

FIGURE 6.34 Class Description: locationDirectoryClass

Figure 6.34 shows the class description for the location directory. The location directory class, much like all the information bases in the CDPD system, consists of a collection of directory entries, which are ultimately bound to the life time of the location directory, and one method for ascertaining the existence of particular location directory entry. Note that the ‘exist’ method is denoted as maximally concurrent, allowing multiple threads of control to be present at any one time.
class **Passive** locationDirectoryEntryClass

attribute locationDirectoryEntryId : locationDirectoryEntryIdType
attribute nei : neiType
attribute rsc : rscType
attribute fs : neiType
method create(nei : neiType, fs: neiType)
method remove()
method **MaxSeq** updateRsc(rsc : rscType)
method **MaxCon** getRsc() : rscType
method **MaxCon** getFs() : neiType

**MutEx**(updateRsc, getRsc)

class end

**FIGURE 6.35 Class Description: locationDirectoryEntryClass**

Figure 6.35 shows the class description for the location directory entry class. The location directory entry contains information on a per NEI basis. Its attributes include the M-ES's NEI, RSC, and FS NEI. The location directory entry contains three methods in addition to its 'create' and 'remove' methods, two maximally concurrent methods for retrieving an M-ES's RSC and FS, and one maximally sequential method for updating an M-ES's RSC. Note that the methods for accessing and updating the RSC are mutually exclusive. Consequently, messages to the 'updateRsc' and 'getRsc' methods must be either synchronous or asynchronous.
class Passive rdrClass
    attribute rsc : rscType
    attribute m-esNei : neiType
    attribute fs : neiType
    attribute authInfo : authInfoType
    attribute locationInfo : locationInfoType
    method create(rsc : rscType, m-esNei : neiType, fs : neiType, authInfo : authInfoType, locationInfo : locationInfoType)
    method route(destinationLus : neiType, sourceLus : neiType)
endclass

FIGURE 6.36 Class Description: rdrClass

Figure 6.36 shows the class description for the RDR class. The RDR class, as with all the classes representing MNLP and MNRP messages, consists of a ‘create’ and ‘route’ method, along with the attributes which form the message. As the ‘route’ method implicitly includes a remove or destroy operation, no synchronization or mutual exclusion information is provided. The RDR message consists of the M-ES NEI, RSC, FS, authentication information, and location information.
Figure 6.37 shows the class description for the RDC class. The RDC class, as with all the classes representing MNLP and MNRP messages, consists of a ‘create’ and ‘route’ method, along with the attributes which form the message. As the ‘route’ method implicitly includes a remove or destroy operation, no synchronization or mutual exclusion information is provided. The RDC message consists of the M-ES NEI, RSC, authentication update information, result code, M-ES configuration timer, and home information.

class Passive rdcClass
    attribute rsc : rscType
    attribute m-esNei : neiType
    attribute authUpdate : authUpdateType
    attribute resultCode : resultCodeType
    attribute configTimer : configTimerType
    attribute homeInfo : homeInfoType
    method create(m-esNei : neiType, authUpdate : authUpdateType, resultCode : resultCodeType, configTimer : configTimerType, homeInfo : homeInfoType)
    method route(destinationLus : neiType, sourceLus : neiType)
endclass

FIGURE 6.37 Class Description: rdcClass

class Passive rdfClass
    attribute rsc : rscType
    attribute m-esNei : neiType
    method create(m-esNei : neiType, resultCode : resultCodeType)
    method route(destinationLus : neiType, sourceLus : neiType)
endclass

FIGURE 6.38 Class Description: rdfClass

135
Figure 6.38 on page 135 shows the class description for the RDF class. The RDF class, as with all the classes representing MNLP and MNRP messages, consists of a 'create' and 'route' method, along with the attributes which form the message. As the 'route' method implicitly includes a remove or destroy operation, no synchronization or mutual exclusion information is provided. The RDF message consists of the M-ES NEI and RSC.

```plaintext
class Passive rdeClass
    attribute rsc : rscType
    attribute m-esNei : neiType
    method create(m-esNei : neiType, resultCode : resultCodeType)
    method route(destinationLus : neiType, sourceLus : neiType)
endclass
```

FIGURE 6.39 Class Description: rdeClass

Figure 6.39 shows the class description for the RDE class. The RDE class, as with all the classes representing MNLP and MNRP messages, consists of a 'create' and 'route' method, along with the attributes which form the message. As the 'route' method implicitly includes a remove or destroy operation, no synchronization or mutual exclusion information is provided. The RDE message consists of the M-ES NEI and RSC.
class **Passive** iscClass

attribute m-esNei : neiType
attribute authUpdate : authUpdateType
attribute resultCode : resultCodeType
attribute configTimer : configTimerType

method create(m-esNei : neiType, authUpdate : authUpdateType,
resultCode : resultCodeType, configTimer : configTimerType)
method route(snpa : snpaType)

class **Passive** mdlpNpduClass

attribute npduBuffer : npduBufferType

method create(npduBuffer : npduBufferType)
method route(snpa : snpaType)

**FIGURE 6.40 Class Description: iscClass**

Figure 6.40 shows the class description for the ISC class. The ISC class, as with all the classes representing MNLP and MNRP messages, consists of a ‘create’ and ‘route’ method, along with the attributes which form the message. As the ‘route’ method implicitly includes a remove or destroy operation, no synchronization or mutual exclusion information is provided. The ISC message consists of the M-ES NEI, authentication update, result code, and M-ES configuration timer.

**FIGURE 6.41 Class Description: mdlpNpduClass**

Figure 6.41 shows the class description for the MDLP NPDU class. The MDLP NPDU class represents a data packet bound from the serving MD-IS to the M-ES. The class consists of the data packet and the ‘create’ and ‘route’ methods. As the ‘route’
method implicitly includes a remove or destroy operation, no synchronization or mutual exclusion information is provided.

```pascal
class Passive msfNpduClass
    attribute npduBuffer : npduBufferType
    method create(npduBuffer : npduBufferType)
    method route(destinationFsa : neiType, sourceFsa : neiType)
endclass
```

FIGURE 6.42 Class Description: msfNpduClass

Figure 6.42 shows the class description for the MSF NPDU class. The MSF NPDU class represents a data packet bound from the home MD-IS to the serving MD-IS. The class consists of the data packet and the 'create' and 'route' methods. As the 'route' method implicitly includes a remove or destroy operation, no synchronization or mutual exclusion information is provided.

6.2.4 Inheritance Graphs

As mentioned previously in Chapter 4, Fusion inheritance graphs are not extended for modelling concurrency during Fusion's analysis and design phases and thus are not included in this case study.
CHAPTER 7
CONCLUSIONS

In summary, this thesis identifies basic object-oriented concurrency modelling requirements by examining existing concurrency modeling techniques. These requirements are then used to form highly integrated concurrency modeling extensions to the Fusion object-oriented development methodology. The Fusion concurrency modeling extensions are then demonstrated using the problem domain of cellular digital packet data (CDPD).

Overall, the object-oriented concurrency modeling extensions to Fusion have proven very capable of modeling the concurrency in a telecom real-time system. They are tightly integrated into both the Fusion notations and process while consistent with the object-oriented paradigm, which is essential to their acceptance.

7.1 Future Research

Several areas of future research are possible from this thesis. First, as mentioned previously, the CDPD problem domain has been scaled down to a manageable size for this thesis. Additional work is required to model CDPD in its entirety to fully discover the usefulness of the Fusion extensions. Also, additional case studies using non-telecom real-time systems (e.g., process control) are required to further test the appropriateness of the concurrency modeling extensions to Fusion analysis and design. The requirement for the assignment and processing of priority messages, scheduling of concurrent processes, and timing constraints are all good examples of possible concurrency modeling which was not required by the CDPD case study but which often occur in real-time systems.

As mentioned previously, Fusion's "implementation" phase includes several steps which are typically included in "low level" or "detailed" designs. The Fusion implementation phase should be extended to provide support for the concurrency extensions presented in this thesis. Of particular concern here might be the inheritance of concurrency properties assigned to base classes. For example, the current extensions to
the analysis and design phases of Fusion do not require that the inheritance of a thread of control be addressed but it could prove to be an interesting topic.

Next, metrics for the comparison of concurrency modeling techniques should be developed and used in a systematic comparison of the extended Fusion methodology and existing methodologies which support concurrency. A single case study should be selected and used for all the selected methodologies to ensure proper comparison the various methods.

Finally, a development language and run-time environment which supports the concurrency constructs presented in this thesis should be selected and used to implement the various designs.


PERMISSION TO COPY

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Texas Tech University or Texas Tech University Health Sciences Center, I agree that the Library and my major department shall make it freely available for research purposes. Permission to copy this thesis for scholarly purposes may be granted by the Director of the Library or my major professor. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my further written permission and that any user may be liable for copyright infringement.

Agree  (Permission is granted.)

__________________________  ____________
Student's Signature        Date

Disagree  (Permission is not granted.)

__________________________  ____________
Student's Signature        Date