Using Simulation for Systems of Systems Hazard Analysis

Robert David Alexander

This thesis is submitted in partial fulfilment of the
requirements for the degree of
Doctor of Philosophy.

University of York
York
YO10 5DD
UK

Department of Computer Science

September 2007

For my parents.

Abstract

For any safety-critical system, thorough and complete hazard analysis must be performed if the system is to be acceptably safe to operate. For the emerging class of systems known as Systems of Systems (SoS), however, performing hazard analysis is extremely difficult because of the complexity of SoS and the environments they inhabit. Traditional exploratory hazard analysis techniques commonly rely upon fixed models of component interaction, and have difficulties exploring the effects of multiple coincident failures. They therefore cannot be relied on to provide adequate hazard analysis of SoS.

This thesis presents a hazard analysis approach that uses multi-agent modelling and simulation to explore the effects of deviant system behaviour within a SoS. A systematic process is defined for developing multi-agent models of SoS, starting from existing models in the MODAF architecture framework and proceeding to implemented simulation models. Throughout this process, a variety of cross-checks between model artefacts provide confidence that the model remains true to the original description and that it adequately describes the SoS being analysed. The exploratory simulations created by the process generate a substantial amount of data concerning the behaviour of the system under a variety of deviations. In order to identify the significant contributory causes of SoS accidents, a tool-supported analysis technique is presented that utilises both machine learning and agent behaviour tracing.

The approach is evaluated against explicit requirements identified for SoS hazard analysis, and through application of the modelling and analysis process to case studies. The case studies demonstrate that the approach can reveal hazards that would be difficult to discover by existing manual hazard analysis techniques.

Contents

Abstract **3**

Acknowledgement **18**

Declaration **19**

1 Introduction **21**

1.1 System of Systems Accidents 21

1.1.1 Blackhawk 21

1.1.2 USS Vincennes 22

1.1.3 Afghanistan GPS 23

1.1.4 Implications of these Accidents 23

1.2 Problems of SoS Hazard Analysis 24

1.2.1 Dynamic Behaviour in Dynamic Structures 24

1.2.2 Difficulty in Deriving the Effects of a Failure 24

1.2.3 System Accidents 25

1.2.4 Presence of Novel Component System Types 26

1.2.5 Requirement for Expertise in Many Domains 26

1.3 Thesis Proposition 26

1.4 Thesis Structure 27

2 Survey of Related Literature **29**

2.1 Systems of Systems 30

2.1.1 What is a SoS? 30

2.1.2 The Motivation for SoS 31

2.2 SoS Engineering 32

2.2.1 The SoS Lifecycle . . . 32
2.2.2 Challenges of SoS Engineering . . . 33
2.3 SoS Safety . . . 34
2.3.1 Safety in Complex Systems . . . 34
2.3.2 Complex Systems, Emergence, and Safety . . . 34
2.3.3 SoS-specific Safety Work . . . 35
2.4 Hazard Analysis . . . 36
2.4.1 The Safety Lifecycle . . . 36
2.4.2 Hazard Analysis Techniques . . . 37
2.5 Automated Hazard and Safety Analysis . . . 42
2.5.1 Automated HAZOP . . . 42
2.5.2 Failure Logic Modelling . . . 43
2.5.3 Monte Carlo Simulation for Air Traffic Safety . . . 43
2.5.4 Friendly-Fire Analysis Using MANA . . . 44
2.5.5 Using SEAS for Collateral Damage Modelling . . . 45
2.5.6 Multi-agent Simulation for Hospital Evacuation . . . 46
2.5.7 SpecTRM-RL . . . 47
2.5.8 Conclusions on Automated Hazard Analysis . . . 47
2.6 SoS Modelling and Analysis . . . 48
2.6.1 Object-Oriented Modelling of SoS . . . 48
2.6.2 DoDAF and MODAF Architectural Models . . . 49
2.6.3 Cellular Automata Models . . . 51
2.6.4 Derivation of DEVS Simulations from DoDAF . . . 52
2.6.5 Generation of Multi-Agent Models from DoDAF . . . 53
2.6.6 Conclusions on SoS Modelling . . . 54
2.7 Conclusions . . . 55

3 Establishing Hazard Analysis as Part of SoS Development 56

3.1 Concepts . . . 56
3.1.1 Defining Characteristics of SoS . . . 56
3.1.2 Contribution of SoS Characteristics to Hazards . . . 58
3.1.3 The Role of Humans in SoS . . . 59

3.1.4 Ownership and Control of SoS . . . 60
3.2 SoS Safety Lifecycle . . . 61
3.3 SoS Hazards . . . 62
3.4 Automating Hazard Analysis . . . 65
3.5 Process Requirements . . . 67
3.5.1 Requirement P1 — Process must be applicable early in lifecycle . . . 67
3.5.2 Requirement P2 — Process must integrate with other safety activities . 67
3.5.3 Requirement P3 — Process must support iterative modelling and analysis 68
3.5.4 Requirement P4 — Process must be feasible and practical to perform . 69
3.5.5 Requirement P5 — Process must provide engineer visibility at all stages 69
3.6 The Core of the Approach — Multi-Agent Simulation . . . 70
3.6.1 Applying Multi-Agent Simulation to SoS Hazard Analysis . . . 71
3.7 Process Overview . . . 72
3.7.1 Role of Human Engineers . . . 73
3.7.2 Role of the Process in the SoS Safety Lifecycle . . . 73
3.8 Running Case Study — Anti-Guerrilla Operations . . . 75

4 Building Simulation Models for SoS Hazard Analysis 77
4.1 Overview of Modelling . . . 77
4.1.1 Choosing What to Model . . . 77
4.1.2 Representing the Real World . . . 78
4.2 Requirements on SoS Simulation Models . . . 80
4.2.1 Requirement M1 — Models must adequately represent SoS . . . 80
4.2.2 Requirement M2 — Models must capture realistic SoS contexts . . . 82
4.2.3 Requirement M3 — Models must incorporate plausible deviations . . . 83
4.2.4 Requirement M4 — Models must provide the features needed for analysis 84
4.2.5 Requirement M5 — Models must be verified and validated . . . 85
4.2.6 Requirement M6 — Models must be amenable to the desired analyses . 86
4.3 A Modelling Process . . . 86
4.3.1 Process Steps . . . 87
4.4 Step 1 — Acquire MODAF Model . . . 88
4.4.1 AGO Case Study . . . 89

4.5 Step 2 — Identify Concerns . . . 89
4.5.1 Relating Concerns to MODAF Model . . . 90
4.5.2 Recording Concerns . . . 91
4.5.3 AGO Case Study . . . 91
4.5.4 Scope and Completeness of Concerns . . . 91
4.6 Step 3 — Identify Missing Information . . . 91
4.6.1 Sources of Additional Information . . . 93
4.6.2 Example of Additional Information . . . 94
4.6.3 AGO Case Study . . . 95
4.7 Step 4 — Decompose Scenarios into Vignettes . . . 95
4.7.1 Generating Vignettes . . . 96
4.7.2 AGO Case Study . . . 97
4.8 Step 5 — Transform Domain Model to Design Model . . . 97
4.8.1 Preservation of Systems of Systems Characteristics . . . 100
4.8.2 Granularity and Level of Detail . . . 100
4.8.3 Verifying the Design Model Against the Domain Model . . . 102
4.8.4 Validating Design Model Against Reality . . . 104
4.8.5 Trace Concerns . . . 105
4.8.6 AGO Case Study . . . 105
4.9 Step 6 — Deviate Design-Level Agents . . . 109
4.9.1 Components of an Agent . . . 110
4.9.2 Guidewords . . . 112
4.9.3 Generic Deviations . . . 112
4.9.4 Deriving Deviation Probabilities . . . 114
4.9.5 Issues of Deviation Independence . . . 115
4.9.6 Reducing the Search Space . . . 118
4.9.7 Deviation Detection and Annunciation . . . 118
4.9.8 Use of Existing Component System-Level Analyses . . . 119
4.9.9 AGO Case Study . . . 119
4.10 Step 7 — Deviate Design-Level Vignettes . . . 120
4.10.1 AGO Case Study . . . 121
4.11 Step 8 — Trace Concerns to Deviations . . . 121

4.11.1 AGO Case Study . . . 122
4.12 Step 9 — Transform Design to Operational Model . . . 122
4.12.1 Choice of Implementation Platform . . . 123
4.12.2 Elaborating the Environment Model . . . 123
4.12.3 Tracing Concerns to the Operational Model . . . 125
4.12.4 AGO Case Study . . . 125
4.13 Step 10 — Implement Operational Model . . . 127
4.13.1 Tracing Concerns . . . 127
4.13.2 Verifying the Implementation Against Domain Model . . . 128
4.13.3 Implementation of Deviations . . . 128
4.13.4 AGO Case Study . . . 128
4.14 Summary . . . 131

5 Analysing the System **133**

5.1 The Problem of Simulation Output Analysis . . . 133
5.1.1 Problem 1 — Size of the Parameter Space . . . 133
5.1.2 Problem 2 — Difficulty of Understanding the Results . . . 134
5.1.3 Problem 3 — Questions About the Validity of the Results . . . 135
5.2 Requirements for an Analysis Method . . . 135
5.2.1 Requirement A1 — Analysis must selectively explore the space provided by deviations . . . 136
5.2.2 Requirement A2 — Analysis must find all paths to accidents . . . 136
5.2.3 Requirement A3 — Analysis must lead to results being validated . . . 137
5.2.4 Requirement A4 — Analysis must present results in human-comprehensible form . . . 137
5.2.5 Requirement A5 — Analysis must present results in terms of SoS mechanisms . . . 138
5.3 A Method for Finding Hazards using SoS Simulation Models . . . 139
5.4 Step 1 — Acquire Model . . . 139
5.4.1 AGO Case Study . . . 140
5.5 Step 2 — Explore the Space . . . 141
5.5.1 AGO Case Study . . . 142
5.6 Step 3 — Learn Hazard Rules . . . 142

5.6.1 Detecting Accidents and Incidents . . . 142
5.6.2 Discovering the Deviations that Lead to Accidents . . . 143
5.6.3 Using Machine Learning to Identify Hazards . . . 144
5.6.4 AGO Case Study . . . 154
5.7 Step 4 — Explain Hazard Rules . . . 156
5.7.1 Using a Tracing Tool to Find Accident Sequences . . . 157
5.7.2 AGO Case Study . . . 163
5.8 Step 5 — Validate Identified Hazards . . . 164
5.8.1 Use of More Detailed Simulations . . . 165
5.8.2 Review by Domain Experts . . . 165
5.8.3 A Caution on Interpretation of Multi-Agent Simulation Results . . . 166
5.8.4 General Questions of Fidelity . . . 167
5.8.5 AGO Case Study . . . 168
5.9 Step 6 — Generalise the Results Across Vignettes . . . 168
5.9.1 AGO Case Study . . . 169
5.10 Step 7 — Feedback to Modelling Process . . . 169
5.10.1 AGO Case Study . . . 169
5.11 Step 8 — Communicate Results . . . 170
5.11.1 Event Trees . . . 171
5.11.2 Fault Trees . . . 172
5.11.3 Commentary — Linearization and Simplification . . . 175
5.11.4 AGO Case Study . . . 175
5.12 Use of the Analysis Results in the SoS Safety Process . . . 175
5.13 Arguing That All Hazards Have Been Identified . . . 177
5.14 Summary . . . 178

6 Case Study **181**
6.1 Adcock Case Study . . . 181
6.2 Phase 1 — Build Model for Adcock Study . . . 182
6.2.1 Step 1 — Acquire MODAF Model . . . 182
6.2.2 Step 2 — Identify Concerns . . . 183
6.2.3 Step 3 — Identify Missing Information . . . 183

6.2.4 Step 4 — Decompose Scenarios into Vignettes . . . 185
6.2.5 Step 5 — Transform Domain Model into Design Model . . . 186
6.2.6 Step 6 — Deviate Design-Level Agents . . . 188
6.2.7 Step 7 — Deviate Design-Level Vignettes . . . 189
6.2.8 Step 8 — Trace Concerns to Deviations . . . 190
6.2.9 Step 9 — Transform Design to Operational Model . . . 190
6.2.10 Step 10 — Implement Operational Model . . . 190
6.3 Phase 2 — Analyse Adcock Model . . . 193
6.3.1 Step 1 — Acquire Model . . . 193
6.3.2 Step 2 — Explore the Space . . . 194
6.3.3 Step 3 — Learn Hazard Rules . . . 194
6.3.4 Step 4 — Explain Hazard Rules . . . 196
6.3.5 Step 5 — Validate Identified Hazards . . . 198
6.3.6 Step 6 — Generalise the Results Across Vignettes . . . 200
6.3.7 Step 7 — Feedback to Modelling Process . . . 201
6.3.8 Step 8 — Communicate Results . . . 201
6.4 Phase 3 — Evaluate Adcock System . . . 202
6.5 Phase 4 — Modify Adcock System . . . 203
6.6 Combined Case Study . . . 204
6.7 Phase 1 — Modelling of Combined System . . . 204
6.8 Phase 2 — Analyse Combined System . . . 205
6.8.1 Step 1 — Acquire Model . . . 205
6.8.2 Step 2 — Explore the Space . . . 205
6.8.3 Step 3 — Learn Hazard Rules . . . 205
6.8.4 Step 4 — Explain Hazard Rules . . . 206
6.8.5 Step 5 — Validate Identified Hazards . . . 207
6.8.6 Step 6 — Generalise the Results Across Vignettes . . . 208
6.8.7 Step 7 — Feedback to Modelling Process . . . 208
6.8.8 Step 8 — Communicate Results . . . 208
6.9 Phase 3 — Evaluate Combined System . . . 208
6.10 Phase 4 — Modify Combined System . . . 209
6.11 Summary . . . 209

7 Evaluation **210**

7.1 Evaluation Against Overall Process Requirements . . . 210

7.2 Evaluation Against Modelling Requirements . . . 213

7.3 Evaluation Against Analysis Requirements . . . 216

7.4 Evaluation Against Thesis Proposition . . . 218

7.4.1 Systematic Development of Multi-Agent Simulations . . . 218

7.4.2 Identification of Emergent Hazards . . . 218

7.4.3 Applicable to SoS . . . 219

7.4.4 Hazards Could Not Easily Have Been Found by Manual Means . . . 220

8 Conclusions **221**

8.1 Summary of Thesis Contributions . . . 221

8.1.1 Requirements for SoS Hazard Analysis . . . 221

8.1.2 Method for Building SoS Simulations . . . 222

8.1.3 Process and Tools for Simulation Hazard Analysis . . . 222

8.2 Future Work . . . 223

8.2.1 Guidance on the Use of More Sophisticated Models . . . 223

8.2.2 Use of Stochastic Models . . . 223

8.2.3 Improved Risk Measures and the Derivation of Risk-Space Models . . . 224

8.2.4 Guidance on Using an Explicit Hierarchy of Models . . . 224

8.2.5 Improved Machine Learning Tools . . . 225

8.2.6 Improvements to Tracing Tool . . . 225

8.2.7 Industrial Trials and Empirical Studies . . . 225

8.2.8 Alternative Run-Selection Heuristics . . . 225

8.2.9 Effectiveness of the Method on Different Classes of SoS . . . 226

8.2.10 Find Influences as Well as Causes . . . 226

8.2.11 Application of the Method to Open SoS . . . 226

8.2.12 Application of Safety and Hazard Analysis to the Method Itself . . . 226

8.3 Overall Conclusions . . . 227

A Despotou's MODAF Model of the AGO SoS **228**

B Revised AGO MODAF Products **234**

C Prometheus Model of the Adcock System **237**

C.1 Goal Overview Diagram 238

C.2 Roles Diagram 238

C.3 Scenario 239

C.4 System Overview Diagram 241

C.5 Agents 242

C.6 Capabilities 246

C.7 Plans 253

C.8 Percepts 253

C.9 Actions 254

C.10 Messages 254

C.11 Data Stores 255

C.12 Protocols 255

C.13 Observations About the Design Model 256

D Deviations for the Adcock System **257**

E Log Output Extract from Adcock Model **269**

Glossary **273**

List of References **277**

List of Figures

2.1 Double-V SoS Lifecycle, Simplified From [1] 33
2.2 V Lifecycle Model Showing Safety Activities, from [2] 37
2.3 Class Diagram of Hypothetical Missile Defence System-of-Systems (from [3]) 49

3.1 SoS Safety Lifecycle Diagram 62
3.2 The Development of Stimuli Into SoS Hazards and Accidents 63
3.3 Overview of the SoS Hazard Analysis Process 72
3.4 The Anti Guerilla Scenario, reproduced from [4] 76

4.1 The Multi-Agent Modelling and Simulation Process (from [5]) 78
4.2 The Information Found in the Design Model (from [5]) 79
4.3 The Phases of the Prometheus Method (reproduced from [6]) 99
4.4 Key for the Prometheus Notation 106
4.5 System Overview Diagram for the AGO System 106
4.6 Agent Diagram for Helicopter Agent 107
4.7 Capability Diagram for 'Attack targets on ground' Capability 107
4.8 The Components of an Agent (reproduced from [7]) 110
4.9 Agent Parts and Services 111
4.10 Filtered Version of the AGO Log Output 130

5.1 An Example of a Log File for a Single Run 135
5.2 The Analysis Process 140
5.3 Decision Tree Learned From the Data in Table 5.1 149
5.4 An Example of an Event Graph for a Simulation Model 157
5.5 An Example Tracing Tool Explanation 161
5.6 A Key to the Tracing Tool Graphs 161

5.7 Tracer Results for Accident 'gh1' . . . 164
5.8 An Example of an Event Tree (from [8]) . . . 171
5.9 An Example of an Accident Fault Tree (reproduced from [9]) . . . 173
5.10 Basic Fault Tree Symbols . . . 174
5.11 Fault Tree for Accident gh1 . . . 176
5.12 Argument That All Hazards Have Been Identified . . . 178
5.13 Argument That the Representation of the SoS is Adequate . . . 179
5.14 Argument That All Important and Plausible Deviations Have Been Identified . 180
5.15 Argument That the Analysis of the Model is Adequate . . . 180

6.1 OV-1a for Adcock System (from [10]) . . . 182
6.2 Revised Version of OV-6c . . . 183
6.3 System Overview Diagram for Adcock System . . . 187
6.4 Initial Trace for UC1I2 . . . 196
6.5 Improved Trace for UC1I2 . . . 197
6.6 Trace for Accident cM1UC1 . . . 198
6.7 Original Version of Fault Tree . . . 202
6.8 Revised Version of Fault Tree . . . 203
6.9 Trace for Accident G1I3 in Combined Vignette . . . 207

A.1 OV-2 Operational Node Relationships Description . . . 229
A.2 Needlines for OV-2 . . . 230
A.3 OV-5 Operational Activity Diagram . . . 230
A.4 OV-6c Operational Event-Trace Description . . . 231
A.5 OV-7 Information Model . . . 232
A.6 SV-4 Functionality Description . . . 232
A.7 SV-5 Function to Operational Activity Traceability Matrix . . . 233

B.1 OV-6c (Operational Event Trace) for AGO SoS . . . 235
B.2 OV-2 (Operational Node Connectivity Description) for AGO SoS . . . 236

C.1 Goal Overview Diagram . . . 238
C.2 Roles Diagram . . . 239
C.3 System Overview Diagram . . . 241

C.4 Agent Overview Diagram for the ‘Infantry’ Agent 242
C.5 Agent Overview Diagram for the ‘Command’ Agent 244
C.6 Agent Overview Diagram for the ‘ISTAR coord’ Agent 244
C.7 Agent Overview Diagram for the ‘MARG’ Agent 245
C.8 Agent Overview Diagram for the ‘MARV’ Agent 245
C.9 Agent Overview Diagram for the ‘UCAV’ Agent 245
C.10 Agent Overview Diagram for the ‘Air Strike Mission’ Capability 246
C.11 Capability Diagram for the ‘Recon Mission’ Capability 247
C.12 Capability Overview Diagram for the ‘BDA Mission’ Capability 248
C.13 Capability Overview Diagram for the ‘Manage MARV’ Capability 249
C.14 Capability Overview Diagram for the ‘Handle BDA Request’ Capability 249
C.15 Capability Overview Diagram for the ‘Ground Assault’ Capability 250
C.16 Capability Overview Diagram for the ‘Self-protection’ Capability 251
C.17 Capability Overview Diagram for the ‘Handle Recon Request’ Capability . . . 251
C.18 Capability Overview Diagram for the ‘Handle CAS Request’ Capability 252
C.19 Capability Overview Diagram for the ‘ISTAR Handle Recon Request’ Capability 252
C.20 Protocol Diagram for the ‘Infantry Recon Request’ Protocol 255

List of Tables

2.1 Steps in FHA (from [2]) . . . 39
2.2 Example of FMEA Output, from [2] . . . 40
2.3 Steps in HAZOP (from [2]) . . . 41
2.4 HAZOP Guide Words (from [2]) . . . 42
2.5 MODAF Products . . . 50

3.1 Contributions of SoS Characteristics to Hazards . . . 59
3.2 Combined Tasks from FHA and HAZOP . . . 65

4.1 Implications of SoS Characteristics for Modelling Requirements . . . 81
4.2 Requirements Implied by Historical SoS Accidents . . . 82
4.3 Guidance for Relating Concerns to MODAF . . . 90
4.4 MODAF-level Concerns Format . . . 91
4.5 Example of MODAF-level Concerns . . . 92
4.6 Categories of Vignette Elements . . . 96
4.7 Representation of SoS Characteristics in Prometheus . . . 101
4.8 Cross-checks Between MODAF and Architectural Design . . . 104
4.9 Cross-checks Between MODAF and Detailed Design . . . 104
4.10 Example of Tracing Concerns into Prometheus-Derived Design Model . . . 109
4.11 Deviation Guidewords, adapted from Pumfrey [2] . . . 112
4.12 Generic Deviations for Agents . . . 113
4.13 Multipliers for Deriving BETA Factor . . . 116
4.14 Agent Deviations for the AGO Case Study . . . 120
4.15 Aspects of Vignettes that can be Deviated . . . 120
4.16 Example of Tracing Concerns to Deviations . . . 122

4.17 Example of Tracing Concerns to Operational Model 126
4.18 Example of Tracing Concerns to Log Output of Implementation 129

5.1 Example Data for a Decision-Tree Learner 149
5.2 MIL-STD-882D Mishap Severity Categories, from [11] 152
5.3 'Frequency of Occurrence' Categories, consistent with MIL-STD-882D 152
5.4 MIL-STD-882D Risk Matrix, from [11] 153
5.5 Possible Accidents 154
5.6 Summary of the Learned Rules 155
5.7 Codes for AGO Deviations 156
5.8 Example of Concepts for a Tracing Tool, from [12] 159
5.9 Example Log for Tracing Tool 160

6.1 Adcock Concerns Against MODAF 184
6.2 Vignette Aspects Derived From Concerns 185
6.3 Adcock Concerns Against Design Model 188
6.4 Adcock Concerns Against Deviations 190
6.5 Adcock Concerns Against Operational Model 191
6.6 Adcock Concerns Against Log Output 191
6.7 List of Deviations Implemented for the Case Study 193
6.8 Top Hazard Rules for Adcock Case Study 195
6.9 Generalised Hazards for Adcock Case Study 200
6.10 Top Hazard Rules for Combined Case Study — Original Version 206
6.11 Top Hazard Rules for Revised Version of Combined Case Study 206

B.1 Needline descriptions for AGO OV-2 235

D.1 Deviations applicable to all agents 258
D.2 Deviations applicable to Infantry agents 259
D.3 Deviations applicable to Command agents 261
D.4 Deviations applicable to ISTAR agents 262
D.5 Deviations applicable to MARG agents 263
D.6 Deviations applicable to MARV agents 265
D.7 Deviations applicable to UCAV agents 267

Acknowledgement

I would like to thank my supervisor, Tim Kelly, for all his advice, guidance and encouragement throughout the development of this thesis.

Much of the work described in this thesis was originally developed for Strand 2 of the HIRTS DARP project. Although the participants of the various DARP Workshops who provided suggestions and criticism are two numerous to name individually, I'd like to thank the Strand 2 working group: Jane Fenn, Andy Ward, Mark Dowding, Alan Wake and Colin O'Halloran.

My two interns, Matt Dickens and Louis Rose, deserve acknowledgement for their invaluable ancillary programming support.

I'd also like to thank all my friends and colleagues (past and present) in the Department of Computer Science, particularly Paul Emberson, Richard Hawkins, Rob Weaver, Rob Collyer, Iain Bate, Mark Coates, Kester Clegg, David White, Ioanna Symeou, Simon Poulding and Peter Laurens. I'd especially like to thank Martin Hall-May and George Despotou for sharing an office with me for an unreasonable amount of time.

Declaration

Some of the material presented in this thesis has previously been published in the following papers:

- R Alexander, M Hall-May and T Kelly, "Characterisation of Systems of Systems Failures" in *Proceedings of the 22nd International System Safety Conference (ISSC '04)*. Providence, RI: Systems Safety Society, August 2004
- R Alexander, M Hall-May, G Despotou and T Kelly, "Towards Using Simulation to Evaluate Safety Policy for Systems of Systems". in *Proceedings of the 2nd International Workshop on Safety and Security in Multiagent Systems (SASEMAS '05)*. Utrecht, Netherlands, July 2005
- R Alexander and T Kelly, "Combining Simulation with Machine Learning to Build Accident Models" in *Proceedings of the 3rd International Workshop on Safety and Security in Multiagent Systems (SASEMAS '06)*. Hakodate, Japan, May 2006
- R Alexander and T Kelly, "Can We Remove the Human from Hazard Analysis?" in *Proceedings of the 24th International System Safety Conference (ISSC '06)*. Albuquerque, NM: Systems Safety Society, August 2006
- R Alexander, D Kazakov and T Kelly, "System of Systems Hazard Analysis using Simulation and Machine Learning" in *Proceedings of the 25th International Conference on Computer Safety, Reliability and Security (SAFECOMP '06)*, ser. LNCS, Janusz Górski, Ed. vol. 4166. Gdansk, Poland: Springer-Verlag, September 2006, pp. 1–14
- R Alexander, M Hall-May and T Kelly, "Ensuring Dependable Systems of Systems" in *Proceedings of the 3rd BAE Systems / SEIC Systems Engineering Research and Technology Conference*. Loughborough, UK, February 2007

Except where stated, all of the work contained within this thesis represents the original contribution of the author.

Chapter 1

Introduction

1.1 System of Systems Accidents

1.1.1 Blackhawk

On 14 April 1994, two Black Hawk helicopters operated by the US Army were shot down by US Air Force F15s, with the loss of 26 lives [13], [14]. The incident occurred during peacetime, in the No-Fly Zone established in Northern Iraq, with no enemy forces in the area.

The two helicopters were carrying out a routine mission, similar to one that they had performed many times before. They were flying that day with incorrect codes set in their Identify Friend-or-Foe[1] (IFF) transponders (they were using the code for the region outside the No-Fly Zone, not the one for inside the zone itself) but this had been the case for several months.

On entering the No-Fly Zone, the two helicopters checked in with the Airborne Warning and Control System (AWACS) aircraft that was controlling the air traffic in the area, and confirmed their identity. A record was made on the AWACS's tracking systems of the helicopters' identity, but this became disconnected from the helicopters when they moved into the radar shadow of the mountains and was never restored to them.

Shortly afterwards, the two F15s entered the Non-Fly Zone to perform a security sweep prior to other air traffic being admitted. They had no knowledge of the Black Hawks' mission (or, indeed of any other Army helicopter activity) and were not expecting to encounter any friendly aircraft.

The F15s eventually detected the helicopters on their radar, and the pilot of the lead F15 'pinged' one of the them with his IFF system, getting a negative (suspect enemy) response, not just with the 'Mode I' system (for which the incorrect code was set) but inexplicably for the 'Mode IV' system that should have functioned regardless of the code that was set.

The wing leader contacted AWACS, seeking to identify the radar blips, but was told that

[1] An IFF system mounted on a vehicle can be remotely queried by another, and will respond with codes that identify the host vehicle type (e.g. military or civilian) and allegiance (e.g. USA)

AWACS had no contacts in that area. Repeating the request a short while later, the AWACS operator told the F15s that they had 'hits' (unidentified contacts) at that location. This was an error: the logs from the AWACS computers show that they in fact had 'paint' — i.e. indicators of entities responding with the friendly (Mode IV) IFF code — at that position.

The F15s carried out a visual identification (VID) pass and the lead pilot identified the Black Hawks incorrectly as Hinds, which were known to be used by the Iraqi forces at the time. His wingman never gave a positive identification. It has been suggested (see [14], [15]) that although the F15 pilots were notionally trained in visual identification, their training was inadequate given the difficulty of the task.

While these events continued, the Black Hawks remained oblivious. Their low altitude operation and limited sensors meant that they had very limited situational awareness with regard to fixed-wing aircraft. More seriously, they were unable to observe any of the radio communication between the AWACS and the F15s because they lacked the advanced 'HAVE-QUICK' anti-jamming radio system that the F15s were using.

At this point, the rules of engagement in force specified that a second VID pass should be performed. Rather than do this, however, the lead F15 shot down the first Black Hawk and his wingman the second.

The accident was able to occur out of combat, in peacetime, in the presence of advanced technological safety measures, airborne surveillance and control, and rules of engagement that required multiple safety checks.

1.1.2 USS Vincennes

On 3 July 1988 the USS Vincennes, an Aegis Cruiser operated by the US Navy, shot down an Iranian airliner with the loss of 290 civilian lives [15], [16].

The Vincennes was on duty in the gulf, as part of a task force protecting shipping from attacks by Iranian gunboats. It was stationed in international waters, and its primary role was to provide air defence for other US vessels.

At the time of the incident a peer vessel, the USS Montgomery, was under attack by small Iranian boats. Despite contrary orders from the task force commander, and in any case being ill-suited to such an engagement, the Vincennes moved to protect the Montgomery, in doing so moving closer to the mainland and (illegally) into Iranian territorial waters.

The Vincennes engaged the Iranian boats with its guns, and had to manoeuvre violently to bring the guns to bear (particularly after one of the forward guns jammed). On the bridge, this created confusion, throwing books and papers around. Whenever the Vincennes fired its guns the lights flickered, further impairing the already poor lighting.

The airliner took off while the Vincennes was still engaged, and the automatic systems on the Vincennes classified it as "Unknown — Assumed Enemy". A crewmember manually triggered the IFF system, and the Vincennes received IFF returns from both the airliner and an Iranian

F14 that was landed at the airport, leading to the assumption that the aircraft was operating in a military role.

As the airliner's course brought it progressively closer, the Vincennes issued several radio warnings, although initially only on military frequencies. Meanwhile, there was confusion on the bridge as to whether the airliner was ascending or descending [16].

A year earlier, the USS Stark had been attacked (accidentally) by an Iranian F14 in a similar situation, with the loss of 37 lives. Craig et al. in [16] suggest that this lead to 'scenario fulfilment' behaviour, with the crew of the Vincennes associating their situation with the (suddenly very salient) memory of the events involving the USS Stark. The scenario may have been particularly compelling because the captain had been forced into retirement as a consequence.

Finally, with the airliner eight nautical miles away, the captain of the Vincennes ordered missiles to be fired, which destroyed it.

1.1.3 Afghanistan GPS

On 5 December 2001, eight US and Afghan servicemen were killed, and over forty wounded, when a US Air Force B52 dropped a laser-guided bomb on a US Army position near Sayd Alim Kalay. The wounded included Hamid Karzai, who had recently been appointed as the leader of the Afghan Interim Authority [17].

The US and allied forces called for an air strike against an identified enemy position, and attempted to give target coordinates based on GPS (Global Positioning System) information. However, in between setting up the enemy position in the GPS system and sending the coordinates to the air controller, the batteries failed in the handheld GPS unit. The GPS operator changed the battery without realising that when restarted the GPS unit always reset its coordinates to its current location.

The headquarters-based air controller coordinated a B52 to attack the specified coordinates, and it carried out the attack accurately.

1.1.4 Implications of these Accidents

The above are examples of Systems of Systems (SoS) accidents — accidents that arise from the interactions of a number of distinct systems within a larger context. The increasing complexity and degree of integration of modern technology, particularly in the aerospace and military sectors, is leading to an increased use of such systems.

It is hard to explain accidents in such SoS, even in retrospect. It is particularly hard to foresee such accidents ahead of time, and to identify the circumstances under which they could occur. An effective risk-based safety process, however, requires that such *hazard analysis* be performed. An inability to do so means that it will be impossible to predict the safety of many future SoS. This may lead to SoS that can be assembled or configured but not deployed or

operated, due to safety concerns, or to accidents in the systems that are deployed.

This thesis is concerned with the hazard analysis of SoS. There is a need for hazard analysis techniques that remain effective in the face of the problems posed by SoS and that are within the skills of typical safety engineers and operations analysts.

1.2 Problems of SoS Hazard Analysis

1.2.1 Dynamic Behaviour in Dynamic Structures

All systems involve some form of dynamic *behaviour*, in that the different parts of the system have state that varies independently over time. Most previous work on hazard analysis, however, has been concerned with systems such as process plants and vehicles which have static *structures*; in such systems, the relationships between the parts are fixed.

SoS, however, may consist of a set of component systems which vary over time, and those component systems may interact in an ad-hoc fashion driven as much by geographical and task-allocation happenstance as by any a priori structure. This follows from some of the characteristics of SoS: geographical mobility of their components, ad-hoc communications between components, and the need for components to coordinate their actions (this list of characteristics is developed further in Section 3.1.1). Hence, the set of interacting component systems, and the relationships between those component systems, varies over time. Any hazard analysis approach that is to be effective when applied to SoS must be able to identify hazards emerging from interactions within such dynamic structures.

It can be noted that the current trend in SoS development is towards increasingly dynamic structures, with the aim of achieving performance or availability benefits. Examples include plans for 'free flight' in civil airspace [18] and decentralised military actions supported by Network-Enabled Capability (NEC) [19].

1.2.2 Difficulty in Deriving the Effects of a Failure

In a conventional system, such as a single vehicle or a chemical plant, the system boundary is well-defined and the components within that boundary can be enumerated. When a safety analyst postulates some failure of a component, the effect of that failure can be propagated through the system to reveal whether or not the failure results in a hazard. The complexity of possible interactions and the large number of possible system states mean that this is not always easy, hence the need for systematic analysis techniques, automated analysis tools, and system designs that minimise possible interactions (e.g. a modular architecture such as that described in Def Stan 00-74 [20]). To make the task more tractable, most existing manual hazard analysis techniques (such as Functional Failure Analysis — FFA — and HAZard and OPerability Studies — HAZOP) deal with only a single failure at a time; coincident failures are rarely considered and systematically explored.

In a SoS, this problem is considerably worse. The system boundary is not well defined, and the set of component systems within that boundary can vary over time, either as part of normal operation (a new aircraft enters a controlled airspace region) or as part of evolutionary development (a military unit receives a new air-defence system). Conventional tactics to minimise interactions may be ineffective, because the SoS consists of component systems that are individually mobile. In some cases, particularly military systems, the component systems may be designed (for performance gain) to form ad-hoc groupings amongst themselves. Consequently, conventional techniques are inadequate in such circumstances for determining whether or not some failure in some component system is hazardous in the context of the SoS as a whole.

It can also be observed that, even if issues of boundaries and dynamic structure could be resolved, SoS components are more complex than their conventional system counterparts. As will be discussed in Section 3.1.1, SoS components exhibit autonomy in pursuit of their own local goals, and this gives them a very complex 'interface' with the system, both in terms of the predictability of their response to immediate stimuli and in the complexity of their response to such stimuli over time.

Autonomous systems (AS) (a category which can be taken, for the purpose of this thesis, to include systems with human operators) can have large amounts of 'mental state' relating to their environment, their assigned tasks, and the activity of their peers. They can respond to that state in a complex, goal-directed way (indeed, their state can include explicit goals, which themselves can change through interactions with peers). How an AS responds to a given stimuli at a given time will be determined by the stimuli received prior to that point, and although this also true for conventional systems, the difficulty of predicting state changes (indeed, even enumerating all the possible states) for autonomous systems is far greater.

1.2.3 System Accidents

Perrow, in [21], discusses what he calls 'normal accidents' in the context of complex systems. His 'Normal Accident Theory' holds that any complex, tightly-coupled system has the potential for catastrophic failure stemming from simultaneous minor failures. Similarly, Leveson, in [22] notes that many accidents have multiple necessary causes. In such cases it follows that an investigation of any one cause *prior to the accident* (i.e. without the benefit of hindsight) would not have shown the accident to be plausible.

A SoS can certainly be described as a 'complex, tightly-coupled system', and as such is likely to experience such accidents. This line of reasoning can be taken slightly further, however, to note that a 'normal accident' could result from actions by each of two component systems that were safe in themselves, but that are hazardous in combination with each other and the wider SoS context.

This latter issue is more immediate when we consider that many SoS exhibit *heterogeneity*; they incorporate systems drawn from multiple manufacturers, developed at different times, and operated by multiple organisations.

1.2.4 Presence of Novel Component System Types

In parallel with increasing interest in developing large, highly-integrated SoS, there has been an increased emphasis on developing autonomous systems (AS) such as Unmanned Air Vehicles (UAVs), largely because of improvements in AS technology. Part of the motivation for increased SoS integration is that such systems depend heavily on support from peer systems due to their limited perception and intelligence capabilities. There is, however, little existing work on the safety analysis of AS, and the complex state-dependency noted above makes it difficult to apply conventional techniques to them.

1.2.5 Requirement for Expertise in Many Domains

SoS are socio-technical systems containing humans, machines, and static structures. They interact with environments containing humans, machines, static structures, geography, weather, animals, and other things. Many of these things can be further subdivided: machines often contain components that are mechanical, electrical, electronic or hydraulic; some components combine all of those (Lisagor et al., in [23], discuss this for civil aircraft systems). Humans, in turn, are studied in a wide variety of distinct fields — psychology, sociology, neuroscience and ergonomics amongst others. SoS, by their nature, span domains of expertise.

It follows from the above that models that capture the system from the perspective of only one domain are not going to adequately describe an SoS in its entirety. Nor will any one human expert be able to adequately understand and analyse the variety of models that are required. Any serious study and analysis of a SoS must involve models that bring in information from multiple domains, and these will need to be produced, studied and used by a diverse range of experts.

Given the need for a diversity of expertise, it is not likely to be useful to present systems and safety engineering methods that take full control of the SoS engineering process in a top-down manner. Rather, innovation in process will need to be piecemeal, with any new methods and processes being explicitly integrated into existing practice.

1.3 Thesis Proposition

The proposition of this thesis is defined as follows:

> It is possible to identify emergent hazards in Systems of Systems, not easily identified using conventional manual hazard analysis, through the adoption of systematic and exploratory multi-agent simulation and analysis using machine learning and agent behaviour tracing techniques.

The following terms from this proposition are worthy of expansion:

emergent hazard a hazard arising from interactions between the component systems of the SoS

systematic and exploratory multi-agent simulation an approach that methodically develops multi-agent simulations from models of the SoS and its environment, and explores the possible behaviour of that simulation to reveal accidents that could happen in the real system

machine learning taking the output from a set of simulation runs where sets of deviations were imposed and simulated accidents occurred, machine learning algorithms can be used to detect relationships between deviations and accidents

agent behaviour tracing techniques when an agent in a simulation is observed to have performed a hazardous action, these techniques allow its decision to perform that action to be traced back to causes in terms of its previous actions, the state of its environment, and the actions of its peers

1.4 Thesis Structure

Chapter 2 reviews the state of the art in the engineering of SoS, including the SoS lifecycle and existing SoS-specific safety techniques. Traditional hazard analysis techniques are surveyed alongside more recent automated analysis techniques, and conclusions drawn as to the adequacy of this work for enabling safety-critical SoS applications.

Chapter 3 shows how hazard analysis can be built into existing SoS development lifecycles. The concept of an 'SoS hazard' is developed to provide the basis for SoS hazard analysis. The role of humans in manual hazard analysis techniques is investigated in order to define the tasks that an effective automated approach must perform. Requirements are then derived for an effective SoS safety process, and a process is outlined that uses multi-agent simulation to address the difficulties inherent to predicting the behaviour of SoS. A case study is presented that is then used as a running example in the following two chapters.

Chapter 4 provides an explicit method for building multi-agent models of SoS for the purposes of hazard analysis. Requirements (derived from the work presented in Chapters 2 and 3) are presented for this use of modelling, and a step-by-step process is presented for producing models that meet those requirements. This process adopts and adapts the Prometheus multi-agent modelling method to assist with the transformation of MODAF models into executable code, and uses a series of identified model stages along with frequent cross-checking of analysis-specified concerns to help ensure that the implemented simulation model is adequate for the hazard analysis task.

Chapter 5 provides a method for performing hazard analysis using the models developed in Chapter 4. As with the two previous chapters, requirements for the analysis method are derived and a method is proposed. It is shown how the combination of existing machine learning techniques combined with an agent tracing tool both allows hazardous combinations of deviations to be identified and allows explanations for that behaviour to be derived.

Chapter 6 describes a case study that was carried out to evaluate the approach. The modelling steps from Chapter 4 and the analysis steps from Chapter 5 are carried out and the results described, with discussion of issues raised.

Chapter 7 evaluates the process described in the thesis against the requirements given in Chapters 3, 4 and 5. It then evaluates the process against the original thesis proposition given in Chapter 1, with particular emphasis on the results of the case study.

Chapter 8 draws conclusions from the work presented in the rest of the thesis, and identifies important and promising future work.

Appendix A reproduces the MODAF model for the running example used in Chapters 3–5

Appendix B presents the author's modifications and improvements to the model presented in Appendix A

Appendix C presents a Prometheus design for the case study used in Chapter 6

Appendix D describes the deviations developed for the case study

Appendix E presents an example of a log produced by running the case study model

Chapter 2

Survey of Related Literature

The application of safety engineering techniques to systems explicitly identified as 'SoS' is a new area, and as such there is little published work specifically concerned with this. Much of the work that *is* now present in the literature was published during the development of the work described in this thesis.

This chapter reviews published work on SoS and particularly SoS safety, along with relevant work from general safety and systems modelling, thereby providing context for the rest of the thesis. It is broken down into sections, specifically:

Systems of Systems Existing definitions and characterisations of the SoS concept, and motivation for their use

SoS Engineering Existing models of the SoS development process

SoS Safety Existing work on the safety problems presented by SoS, and the nature of techniques that could resolve those problems

Hazard Analysis Established manual techniques for performing hazard analysis on a variety of systems

Automated Hazard and Safety Analysis Existing techniques for performing automated hazard analysis that are directly applicable to SoS or could feasibly be adapted to this purpose

SoS Modelling and Analysis Existing notations and methods for performing modelling and analysis of SoS for purposes other than safety

2.1 Systems of Systems

2.1.1 What is a SoS?

The term 'Systems of Systems' remains controversial, and there are several competing definitions in use. In part, this stems from competing intuitive ideas as to what does and does not constitute an SoS. For example, most researchers and practitioners would agree that a network-enabled military force is a SoS, but for the case of a distributed engine control system opinion would be divided. The US Department of Defense Defense Acquisition Guidebook [24] suggests that a combat aircraft is "an example of a system of systems", while Pyster and Gardner, in [1], explicitly reject this statement.

Maier [25] notes that *"while the term 'systems-of-systems', has no clear and accepted definition, the phenomena is widespread and generally recognized."*

The term has been used by the US Department of Defense to describe the proposed Network-Centric Warfare concept [26], which is discussed further in Section 2.1.2. This is the context in which the term is most frequently encountered.

Kotov, in [27], defines SoS to be *"large-scale concurrent and distributed systems, the components of which are complex systems themselves"*. His examples include *"multiprocessor servers and clusters"* and *"distributed control systems"*. There is no mention of large systems such as NCW and controlled airspace, although they are not explicitly excluded. This emphasis is not uncommon — although the work that this thesis relates to is primarily motivated by large-scale military and transport systems, many researchers studying 'SoS' are concerned with distributed information and business systems (often described as 'enterprise networks').

In [28], the Dependable Systems of Systems project at the University of Newcastle proposed the following definition of an SoS:

> *"The component systems of an SOS will in general be existing systems, comprising hardware and software, each potentially under separate management and ownership. Due to the autonomy of the individual systems, an SOS is not just a large complex distributed system, but rather, one whose component systems:*
>
> - *fulfil valid purposes in their own right and continue to operate to fulfil those purposes if disassembled from the overall system, and*
> - *are managed (at least in part) for their own purposes rather than the purposes of the whole."*

Whereas Kotov considers it enough that a SoS is formed from several systems, this definition adds the requirement that the component systems have functions beyond that of the enclosing SoS. This is significant in that it encompasses many purported examples of SoS while excluding others. For example, a distributed engine control system is not a SoS, by this definition, because its components have no purpose or 'life' beyond the purpose of the system.

Although the military and transport systems would be covered by the Newcastle definition, there is no specific coverage of them in the published work [29]. It would appear from the assumptions implicit in the text that such systems have not been considered.

Periorellis, in [30], states that *"A DSoS [Dependable System of Systems] is a dependable system composed of independent autonomous systems. The purpose of a SoS is to provide a set of enhanced or improved 'emergent' services, based on some or all of the services provided by the participating component systems. The provision of these emergent services requires co-operation between the systems."*

Periorellis' definition clearly encompasses Kotov's, and the phrase 'independent autonomous systems' can be taken to cover the independence proviso in the Newcastle definition. Periorellis has also added the concept of emergent services, and the requirement that they are somehow 'enhanced or improved'. This latter point excludes those systems in which the component systems are brought together only for some administrative or organisational purpose, and where the component systems work only independently of the others.

Maier [25] takes a slightly different approach, proposing five characteristics that distinguish SoS from large monolithic systems:

> *"Systems-of-systems should be distinguished from large but monolithic systems by the independence of their components, their evolutionary nature, emergent behaviors, and a geographic extent that limits the interaction of their components to information exchange."*

This definition includes all of Periorellis', with the addition of 'evolutionary nature' and 'geographic extent'. It is notable that Maier's 'geographic extent' seems to exclude military and transport systems, which by their very nature can interact physically as well as by information exchange.

The characteristics of SoS, as given by the definitions above, lead to the problems for hazard analysis presented in Section 1.2. For example, Periorellis' 'emergent' services may occur in tandem with emergent hazards that existing hazard analysis techniques are ill-equipped to predict. Such 'negative emergence' is a major concern for SoS safety.

2.1.2 The Motivation for SoS

Given the problems presented by SoS, as discussed above and in Section 1.2, why is there continued interest in building and operating such systems? It can be observed that the title of this section is somewhat misleading; Systems of Systems have always existed, and there are many large examples including the Internet and the various Air Traffic Control systems that are in operation around the world. That such systems are SoS is an inherent part of their nature; they could not be replaced by anything that was not.

It can be observed, however, that with these existing SoS, there is a trend towards increased

integration, complexity, flexibility and dependence on emergent behaviour. These trends compound the problems that motivate this thesis.

As noted above, the most prominent example of SoS-by-design is the move towards Network Centric Warfare in military operations. Alberts [31] offers *"NCW is an approach to the conduct of warfare that derives its power from the effective linking or networking of the warfighting enterprise. It is characterized by the ability of geographically dispersed forces (consisting of entities) to create a high level of shared battlespace awareness that can be exploited via self-synchronization and other network-centric operations to achieve commanders' intent."*

In essence, NCW is an approach to warfare whereby all parts of the military force share information via a network. In theory, every element of the force has access to the information acquired by all others — effectively, their entire sensing capability is pooled.

For example, if a reconnaissance plane observes a hostile vehicle in a valley, this information is immediately available, via the network, to the theatre command post in a nearby town, to the ground-attack aircraft circling nearby and to the infantry platoon on the other side of the hill. All of these elements can now respond immediately to the new information.

In the United States military, NCW is seen as the centre of the Revolution in Military Affairs (RMA), which is the coming together of military technology and organisations with commercial advances in Information Technology and information systems [31]. In the UK military, the term Network Enabled Capability is more commonly used [19], but the core concept is the same and has the same qualitative properties with respect to safety.

It should be noted that although there is widespread advocacy for NCW, there is little empirical research to support the claims for its effectiveness and viability, and there is a prevalent aspect of hyperbole to much of the US military output on the subject. It is true, however, that the general theme of decentralised operations has a strong pedigree in military studies over the last century, with those nations that maintained overly centralised structures being seen to be at a disadvantage in a number of conflicts [32].

Regardless of the actual value of NCW or NEC approaches, such systems are being developed and there is a need for safety engineering techniques to support them.

2.2 SoS Engineering

There is now substantial literature on SoS engineering in general, but little of it is directly concerned with the subject of this thesis. The literature often presents interesting discussions of problems, but frequently falls short of providing concrete solutions.

2.2.1 The SoS Lifecycle

Pyster and Gardner, in [1], present a candidate SoS lifecycle based on a 'double-V' concept — a large outer 'V' for the whole system, and then a set of inner 'V's, one for each component

system. This lifecycle is depicted in Figure 2.1.

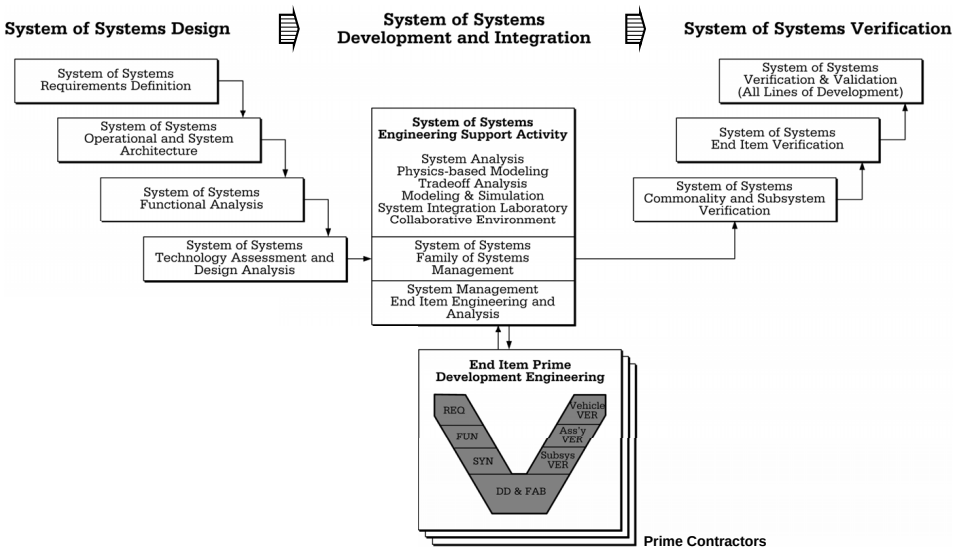

Figure 2.1: Double-V SoS Lifecycle, Simplified From [1]

It can be observed that the lifecycle for the development and procurement of the overall SoS feeds into that of the individual systems ('End Items' in the diagram), and vice-versa. This will inevitably impose requirements on both sides — the engineers controlling the SoS development must provide the system developers with the information they need to develop and evaluate their systems, while the system developers must provide the SoS developers with ongoing feedback as to how their system will actually operate so that the SoS engineers can analyse and control the development of the whole SoS.

Keating et al., in [33] state that for SoS *"it is naïve to assume that 'one size fits all' and a singular n-step process can be successfully applied to any complex system problem context with an equivalent probability of success"*. They propose, instead, that the focus be on engineering 'methodologies' as defined by Checkland in [34]: *"... not* a method *but a set of* principles of method *which in any particular situation have to be reduced to a method uniquely suitable to that particular situation."*. This has implications for the value of any explicit SoS lifecycle or engineering process — inevitably, steps or phases will be performed in an order or with repetitions that cannot be easily forseen in advance.

2.2.2 Challenges of SoS Engineering

It is widely recognised that SoS engineering is difficult. For example, Keating et al. in [33] note that *"traditional systems engineering has not been developed to address high levels of ambiguity and uncertainty in complex systems problems"* and that *"For future complex systems, it is naïve to think that problem definitions and requirements will be isolated from shifts and pressures stemming from highly dynamic and turbulent development and operational environments."*

One widely recognised problem is that of the potential for coupling between apparently very distinct component systems in the SoS. Raheja and Moriarty, in [35], observe that *"Systems today are connected directly and indirectly with many other systems. A typical weapon system is linked to other systems, vehicles, soldiers and satellites in almost unlimited interactions. Tweaking in one place is bound to create some change in another place."*

Similarly, the Department of Defense Defense Acquisition Guidebook [24] states *"The development of a system of systems solution will involve trade space between the systems as well as within an individual system's performance."* It goes on to note that individual system developers must therefore be aware of how the characteristics of SoS could impact on the necessary architecture of their system and on how it delivers its capabilities, and provides a checklist for system developers prompting them to consider a variety of integration issues.

It follows from the above that repeating analysis frequently is a necessary and inevitable part of SoS engineering. Independent development and simply-structured integration (whether top-down or bottom-up) will not be possible. Iterative models of development will be necessary.

A final problem is the need for what Keating et al., in [33], call *"whole systems analysis"*. SoS cannot be adequately analysed or engineered from a purely technical perspective — the human dimension (from technical device operators through to the organisational and political levels) must be thoroughly treated. Keating et al. suggest that current SoS engineering work overemphasises the technical at the expense of these other factors.

2.3 SoS Safety

2.3.1 Safety in Complex Systems

As noted in Section 1.2.3, Perrow [21] claims that any complex, tightly-coupled system has the potential for catastrophic failure stemming from simultaneous minor failures. Further, he claims that complex systems inevitably experience frequent minor failures, and that engineers will never be able to anticipate all possible combinations of failure.

Although Perrow discusses the causes of normal accidents, and the aspects of system complexity that make them likely, he does not offer any approaches for reducing their occurrence beyond the simple suggestion that one should build simpler systems.

2.3.2 Complex Systems, Emergence, and Safety

There is considerable current interest in the field of emergent behaviour in complex systems. Much of this has its roots in systems theory, although there is also the increasingly popular (though often nebulous) field of 'complexity theory'.

From a systems theory perspective, Checkland [34] defines emergence as *"The principle that whole entities exhibit properties which are meaningful only when attributed to the whole, not*

to its parts — e.g. the smell of ammonia. Every model of a human activity system exhibits properties as a whole entity which derive from its component activities and their structures, but cannot be reduced to them."

The subject of emergent behaviour has seen little thorough coverage from a scientific or engineering perspective. There are numerous popular science texts on the subject, which discuss the qualitative issues and impart an intuitive grasp of the subject (e.g. [36, 37]), but they stop short of offering concrete and falsifiable theories.

Resnick, in [38], discusses human understanding of decentralised systems, although his focus is on children rather than adults. He discusses the use of simulated emergent behaviour as a guide to this. His examples illustrate some of the difficulties in understanding emergent behaviour. For example, in a simple simulation of a one-lane road, an apparently persistent 'traffic jam' entity can be observed, even though the set of cars that it contains is constantly changing. The difficulty of comprehending decentralised phenomena from static models suggests that simulation may be important for SoS analysis.

2.3.3 SoS-specific Safety Work

The Dependable Systems of Systems research project at the University of Newcastle explored many aspects of dependability in Systems of Systems. As noted in Section 2.1 they used a definition of a SoS that would be widely recognised as such. Their initial 'Description of Work' states:

> *"The overall objective of the DSoS project is to develop significantly improved means for composing a dependable system of systems (SoS) from a set of largely autonomous component computer systems."* [39]

Note the specific use of "component computer systems" — the project's work was concerned with enterprise networks and business information systems. The military and transport systems that this dissertation is concerned with were *not* included, although this was never made explicit in the published reports. Such a bias towards information technology concerns is common in the extant SoS engineering work, as observed by Keating et al. in [33].

It follows that the DSoS researchers had no need to consider safety explicitly in their research, and their definition of dependability seems to reflect this bias:

> *"Dependability: The dependability of a system is the ability to deliver a service that can justifiably be trusted, where the service is the intended behaviour of the system."* [29]

As noted in Section 2.2.2, the complex interactions between elements of a SoS mean that changes to one element may necessitate changes to many others that are apparently quite distinct. Raheja and Moriarty, in [35] propose that by way of a solution *"System safety needs*

to pay more attention to hazard analysis on the structure and architecture of the system-of-systems. The architecture must have traceability to safety-related interactions.". In other words, safety analysis must be performed on the SoS as a whole, and starting from very early, high-level models.

Leveson, in [40] notes that traditional safety analysis techniques make assumptions that are highly applicable in the domains for which they were originally developed (typical, simple mechanical and automotive systems), but are inappropriate for complex modern systems and SoS. One such assumption is that accidents usually result from distinct failures of a component; as Perrow also noted (see above), in complex systems it is often the interaction of apparently acceptable and normal behaviour that leads to accidents.

Leveson therefore proposes that SoS safety engineering should be based on new model, which she calls Systems-Theoretic Accident Modelling and Processes (STAMP). In STAMP, the basic model element is not the entity but the constraint, and accidents are seen as emerging from the failure of safety constraints. The basic STAMP modelling notation is very flexible and STAMP models can extend well beyond the physical system under analysis and incorporate operators, owners and the legislative environment.

STAMP can be used as the basis for SoS accident analysis. For example, in [13] Leveson presents an analysis of the Black Hawk friendly fire accident described in section 1.1.1. However, Johnson, in [41], notes that although Leveson proposes that the STAMP method be used for accident analysis, she provides no process for doing this, and when he performed a study with two analysts performing separate analyses of one accident, the two STAMP models they produced were significantly different.

2.4 Hazard Analysis

2.4.1 The Safety Lifecycle

Pumfrey, in [2], presents a representation of the system safety lifecycle in the form of a 'V' model, derived from the V-diagram for software development as presented by McDermid and Rook in [42]. Pumfrey notes that it is a useful framework to use when describing the different roles of various analysis techniques.

Figure 2.2 is a depiction of the V model, taken from Pumfrey. The left-hand peak of the 'V' is the start of the process, beginning with requirements analysis. At this stage only Preliminary Hazard Identification (see Section 2.4.2.1) can be conducted. As the process proceeds down the left edge, the design becomes progressively more detailed, and it is used to feed predictive hazard analyses such as Functional Failure Analysis (see Section 2.4.2.2). Having a detailed design also makes it possible to find common-cause failures, using techniques such as Zonal Hazard Analysis. The bottom of the V represents implementation.

Moving up the right-hand side, development moves into system integration and testing, verifi-

cation and validation. Techniques such as Fault Tree Analysis and Failure Modes and Effects Analysis (FMEA) can be used to confirm that the safety requirements identified during the earlier phases of development have been satisfied. The right peak of the V culminates with delivery of the completed system and its associated safety case.

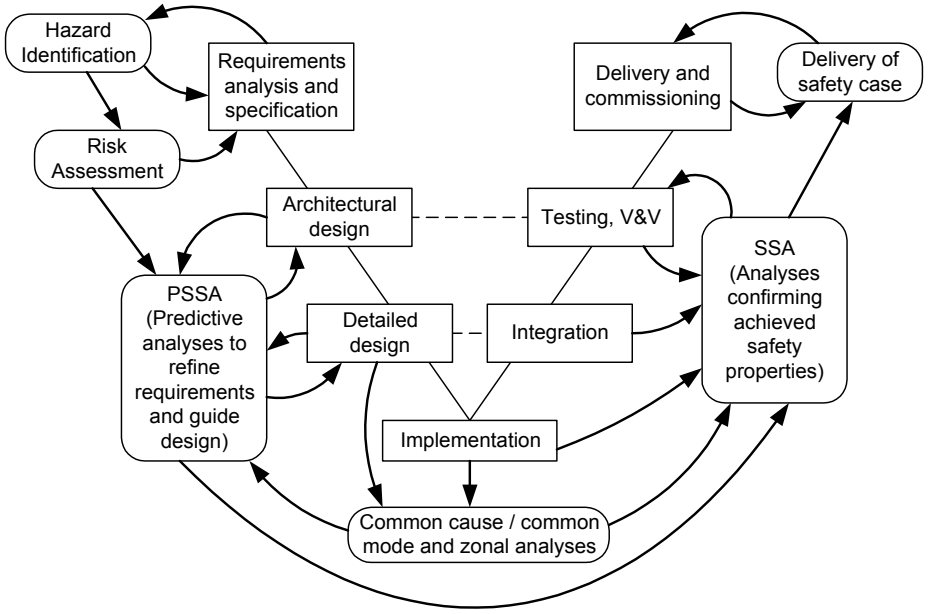

Figure 2.2: V Lifecycle Model Showing Safety Activities, from [2]

Hazard and safety analysis techniques have distinct roles within the lifecycle. Hazard analysis is primarily inductive, working forwards from a high-level description of a system to enumerate possible hazards. By contrast, safety analysis is usually deductive, working backwards from hazards, via a detailed design, to find ways in which those hazards could occur, and to quantify risks.

As noted in Section 2.2, SoS complicate the lifecycle. Their 'design' stages are often not design so much as procurement, assembly or formation of existing resources. During this process there is a need to analyse the safety of the component systems that are being brought together, but this is only possible given some knowledge of the overall system context.

Whatever the mapping between conventional system design and its SoS equivalent, SoS need to be subjected to a systematic and thorough hazard analysis. The following section reviews several techniques that may be useful for achieving this.

2.4.2 Hazard Analysis Techniques

A huge range of hazard and safety analysis techniques are described in the literature — the Systems Safety Analysis Handbook [43] lists 101 distinct techniques, and it is not exhaustive. Admittedly, many of the techniques covered are highly specialised; Bent Pin Analysis deals

exclusively with failures in cable connections, and Nuclear Safety Analysis is only relevant to nuclear reactors.

The concern in this thesis is exclusively with hazard analysis techniques, primarily what Fenelon et al. [44] describe as *exploratory* analysis — the starting point is that causes and effects are unknown, and the desired result of the analysis is that plausible causes and effects have been derived. *Inductive* techniques, which derive an unknown effect from a known cause, are also of interest, but are insufficient on their own without exploratory support.

In this section, several common hazard analysis techniques will be reviewed, considering their applicability to SoS. The criteria for selecting these few techniques are:

- They must be either *exploratory* or *inductive*
- They must be applicable early in the lifecycle
- If they are not SoS-specific, they must have potential for application to SoS

2.4.2.1 Preliminary Hazard Identification (PHI)

Pumfrey, in [2], describes PHI as being a process of brainstorming for possible hazards, followed by critical review of the hazards derived. The intent is to exploit past experience to predict hazards that are likely to occur in a new system. PHI approaches typically use aids such as checklists that incorporate a wide range of problems that have previously been encountered or proposed.

PHI techniques are potentially applicable to any type of system, but their value in the SoS context is limited by the novelty of many of the SoS concepts that are now being proposed.

2.4.2.2 FFA/FHA

Functional Hazard Analysis elicits hazards from a functional description of a system. It is an inductive technique, working forwards from a description of a system function to discover ways that the failures of the function could cause or contribute to unsafe behaviour.

In FHA, the analyst starts with a description of a system's external functions, and for each function they identify the ways in which it can fail (the failure modes). The effects of each failure are then considered, based on the information available about the system at that time. This then provides a basis for discussing improvements to the design, with the intention that the identified hazards be mitigated, or prevented from occurring. The steps of the FHA process, as described by Pumfrey in [2], are shown in Table 2.1.

FHA is commonly performed in a variant known as Functional Failure Analysis (FFA). In this, specific types of failure are considered for each function: 'function not provided', 'function provided when not required' and 'function provided incorrectly'.

Step	Description
1	Identify functions.
2	For each function identified, suggest failure modes, using the three failure types as prompts.
3	For each failure mode, consider the effects (at different levels, e.g. function / subsystem / system, if necessary).
4	Identify and record any actions necessary to improve the design.

Table 2.1: Steps in FHA (from [2])

Pumfrey [2] notes that *"FFA is a predictive technique, which should be applied early in the development of a design to help identify and refine safety related requirements."*.

It is notable that FHA/FFA can be performed on a set of functions alone, without reference to their specific implementations. For a SoS, this could mean that FFA could be conducted over the functional requirements for a SoS concept, before the specific system configuration for the SoS had been decided on. This analysis early in the lifecycle is potentially very valuable. For a System of Systems, FFA is problematic in that it is concerned with only a single failure at a time. It follows that 'system hazards', where multiple failures contribute to a single hazard, are unlikely to be identified by FFA.

Dealing with the many possible configurations of a SoS is also problematic, as the result of a particularly functional failure will depend heavily on the system context. Some failures will only be of consequence in a one of many system contexts, and some events may be completely safe in isolation (when the component system is viewed as an isolated entity), but hazardous when placed in a particular SoS context. With the dynamic reconfiguration exhibited by many SoS, this is a serious problem.

2.4.2.3 FMEA

In FMEA, the system to be analysed is broken down into components, and known failure modes of those components are used to derive possible consequences at the system level. It can be noted that FMEA is not a single technique, but rather a family of related techniques [2, 8]. Aerospace Recommended Practice 4761 [45] describes two versions, one taking a functional approach and the other based on components and their composition.

Typically, FMEA involves building a table in which system components are listed. For each component, the known failure modes are identified, and their effects on the system as a whole are detailed. Variations may expand on this with additional columns, such as a discussion of contributory factors, or means of detecting or mitigating the failure. Table 2.2 shows an example of the resulting table.

FMEA could reasonably be applied to SoS. The analyst in FMEA can define 'component' to mean whatever is appropriate for the system under analysis — in the case of an air transport SoS it the components could be large-scale entities such as aircraft, radar installations and air traffic control centres. Compared to FFA, FMEA is applicable later in the lifecycle, because of the need to have previously identified the system's components and their failure modes.

Component	Failure Mode	Subsystem Effects	Vehicle Effects	Comments
Variable Reluctance Sensor	No signal	Vehicle speed will always be calculated as zero	1) No speed indication 2) Mileometer not incremented 3) Electronic gearbox control may select too low a gear, possibly resulting in wheel lockup or transmission damage	Effect (3) requires simultaneous failure of engine load calculation and mechanical interlocks on gearbox
Variable Reluctance Sensor	Noisy (too many edges)	...	...	...

Table 2.2: Example of FMEA Output, from [2]

For use with SoS, FMEA shares with FFA the difficulty of deriving the effect that a given component failure mode will have at the SoS level, and the standard procedure only considers the effect of a single failure at a time. Similarly, considering multiple SoS configurations will require multiple analyses.

2.4.2.4 HAZOP

HAZard and OPerability Studies (HAZOP) [46] was developed in the chemical industry for application to chemical plants. It is primarily concerned with flows between components, looking for the effects of possible deviations in the behaviour of these flows.

The steps of the HAZOP process are shown in Table 2.3. The core of the process is the use of guide words to prompt analysts to consider a variety of possible deviations for each flow. Pumfrey [2] notes that the set of guide words used in industry is fairly stable, and offers Table 2.4 as a list and explanation of these.

For each deviation, the analyst can consider possible causes and consequences. As such, the method has both inductive and deductive elements. Like FFA, however, it requires the inductive analysis to be completed if the technique is to work at all — if the effects of deviations cannot be predicted, then the process cannot continue.

HAZOP can be applied to SoS, but it is not immediately obvious what the analogue to 'flows' would be. Possibilities include information flows, shared resource relationships, and collaboration interactions.

In a SoS context, HAZOP has similar problems to FFA - the set of components and flows will change over time as a result of dynamic collaboration and SoS evolution, and this will invalidate the existing HAZOP analysis. Again, conducting effective analysis requires knowledge of the SoS context, and (unlike the piping structure of a typical process plant) this isn't stable during the operational life of the system.

Step	Description
1	Select a flow in a pipeline.
2	Identify important physical attributes of the flow, such as *pressure, temperature, flow rate, chemical composition* etc.
3	Consider those deviations prompted by applying each guide word to each property for this line section.
4	Determine the possible causes of each of these deviations.
5	Investigate the expected outcome (effect on the plant) of each deviation, taking into consideration operating conditions and other causal factors where necessary, and examining the contribution of protection mechanisms and other mitigation already included in the design.
6	Decide which deviations are safety problems (i.e. those with both plausible causes and hazardous effects).
7	For deviations which are not safety problems, record a justification (i.e. explain why the design is acceptable as proposed.
8	Consider changes to the plant that will remove, or reduce the probability or severity of, hazardous deviations.
9	Determine whether the cost of the proposed changes is justified.
10	Agree actions and responsibilities.
11	Repeat steps 1 to 10 for all other lines in the plant.
12	Follow up to ensure necessary actions have been taken.

Table 2.3: Steps in HAZOP (from [2])

2.4.2.5 STPA

STPA (STamP Analysis) is a hazard analysis technique based on the STAMP accident model (see Section 2.3.3), and is described by Leveson in [47]. Leveson notes that *"Most hazard analyses (with the exception of HAZOP) are performed on system models contained only in the minds of the analysts. In contrast [STPA analyses] use concrete models of the process."*

The STPA process proceeds from a set of hazards defined at the whole-system level, and derives from this appropriate safety requirements and safety constraints. A system dynamics model of the system is then used to derive 'inadequate control actions' (actions that could cause the safety constraints to be violated) via a standard set of 'control loop flaws' (these have a similar role to the guide words used in HAZOP). Design changes are then proposed to prevent the inadequate actions occurring.

STPA is clearly *intended* for SoS analysis. It is not clear from the published work on STPA, however, whether it scales well to large systems with many component systems and complex control structures. As a manual technique, there is also the question of whether it can efficiently consider multiple SoS configurations, environments and missions.

2.4.2.6 Conclusions About Hazard Analysis Techniques

The manual methods described in this section are potentially applicable to SoS, but it is likely that they will perform badly in the face of the characteristics of SoS. There is, therefore, a need for new approaches.

Guide Word	Deviation	Example Interpretation
NO or NONE	No part of the intention is achieved	No forward flow when there should be
MORE	Quantitative increase in a physical property (rate or total quantity)	Higher pressure, flow rate, temperature... Quantity of material is too large.
LESS	Quantitative decrease in a physical quantity (rate or total quantity)	Lower pressure, flow rate, temperature... Quantity of material too small
MORE THAN or AS WELL AS	All intentions achieved, but with additional effects (qualitative increase)	Impurities in flow (air, water, oil...), chemicals present in more than one phase (vapour, solid)
PART OF	Only some of the intention is achieved (qualitative decrease)	One or more components of mixture missing, or ratio of components is incorrect
OTHER THAN	A result other than the intention is achieved	Any state other than normal operation, e.g. startup, shutdown, maintenance...
REVERSE	The exact opposite of the intention is achieved	Reverse flow

Table 2.4: HAZOP Guide Words (from [2])

2.5 Automated Hazard and Safety Analysis

One solution to the problems with manual analysis methods is to attempt to automate the analysis process. Automated hazard analysis potentially allows the exploitation of modern computing power to consider huge numbers of combinations of failures, SoS structures, and environments, and to find any hazards that are exhibited.

Pumfrey [2] criticises typical automated tools for being unable to judge the criticality of hazards identified, thereby forcing users to manually review large amounts of output. It is clear that a tool is of little use if the time taken to analyse its output equals or exceeds the time taken to perform the tool's job by hand. With SoS, the potential for hazard analysis to be excessively time-consuming is significant. This is therefore a crucial factor in the viability of automated techniques.

2.5.1 Automated HAZOP

The area of automated hazard analysis that is perhaps best represented in the literature is that related to automated HAZOP. Venkatasubramanian et al. present a summary of this work in [48].

For example, QHI (described by Catino and Ungar in [49]) combines a plant-specific pipe and

instrumentation diagram of a chemical plant with a generic library of faults, chemical processes and hazards. Permutations of the possible faults are combined with the plant model, producing modified versions of that model, and the behaviour of these variants is then simulated using a qualitative simulation approach (as described by Kuipers in [50]). The QHI tool then looks for behaviour that is consistent with the generic set of hazard types, and reports such behaviour to the user, along with the faults that caused it.

The existing automated HAZOP work is successful in part because it exploits the highly static structure and limited domain of chemical process plants. As noted above in Section 2.4.2.4, this assumption does not hold for SoS. The work on automated HAZOP is therefore not directly applicable to SoS. The lack of an obvious SoS equivalent of the Pipe and Instrumentation diagram is also an issue, although potentially such an analogue could be proposed, at least for partial analysis (for example, over information flows).

2.5.2 Failure Logic Modelling

There is a body of work on what Lisagor in [23] terms 'failure logic modelling' — an approach to predicting failures at the interface of a system by studying how discrete 'failures' can enter the system, or be generated internally, and then using a model of failure propagation to determine how they propagate through the system. Examples of this include Papadopoulos et al. on Automated FMEA [51] and by Wallace's Fault Propagation and Transformation Calculus [52].

Failure logic modelling is primarily an inductive analysis. It can be considered to be exploratory, in the sense that it derives system-level behaviour from descriptions of component-level failure behaviour.

The difficulty in applying such an approach to SoS is that it presupposes that we can circumscribe the failure propagation behaviour of autonomous agents within a SoS. Such approaches also typically assume a predefined sequence of interactions, over statically defined channels (often defined by the composition of components). The emphasis on modelling failures is also a limitation when considering negative emergent behaviour that can arise from the interaction of normal behaviours of component systems. In SoS, we require an approach that models normal behaviour but includes the potential to examine the effects of failures on the normal behaviour and interactions of component systems. (Such an approach would be such more akin to fault injection on a working system — see, for example, Voas in [53]).

2.5.3 Monte Carlo Simulation for Air Traffic Safety

Blom, in [54], describes an approach to the analysis of complex safety critical systems that uses Monte Carlo simulation. It is based around the TOPAZ (Traffic Organisation and Perturbation AnalyZer) 'safety risk assessment methodology' (described by Blom [55]), and the simulation element comes into play in the later stages of that process. Once a manual analysis has been performed to define the system under analysis, identify the hazards that are believed to exist

in the system, and to identify scenarios that need to be considered, a simulation model is built to calculate the risk of each hazard occurring. Monte Carlo analysis is used to get statistically significant results from the simulation.

The TOPAZ method is intended for confirmatory safety analysis, but the use of multi-agent simulation can assist the later stages of hazard analysis by exploring how the various stochastic factors in the model lead to accidents occurring. However, the scope of the situations explored in the simulation is driven by a prior hazard analysis at a relatively low level — an example of a scenario to be explored is given in [54] as *"Aircraft erroneously in takeoff and crossing aircraft on runway"*. The method does not, therefore, help with exploratory analysis as described by Fenelon (see Section 2.4.2).

Furthermore, the main output of the method is a set of statistical safety measures. This is a necessary output from a safety analysis process (it is valuable evidence for building a safety case), but it is of limited use to engineers who need to understand *how* the system reaches a hazardous state, which is a necessary prerequisite to preventing hazards occurring or mitigating their effects.

Additionally, the usefulness of such purely statistical information depends heavily not just on the fidelity of the simulation but on the ability of the engineer to support that claim of fidelity. Hence, in hazard analysis, engineers need something that they can use as a starting point for investigation; they need results that they can use, not results that they have to rely on. Dewar et al., in [56], call this "weak prediction".

The example presented in [54] is of a runway where multiple aircraft periodically need to cross it, while at the same time the runway is used for takeoff and landing. The example is small, consisting of a few aircraft interacting in a single runway and surrounding aprons and taxi lanes. This, combined with the computational load imposed by the Monte Carlo approach, raises doubts about the scalability of the process to large, complex systems.

2.5.4 Friendly-Fire Analysis Using MANA

MANA (Map-Aware Non-uniform Automata) is a cellular automata model used for modelling military performance, described by Lauren and Stephen in [57]. Automata representing military units move about a flat plane that represents the terrain, performing mission goals (usually expressed as navigation between simple waypoints) and engaging in combat with enemies they encounter. Their behaviour is controlled by the assigned waypoints and a set of tendencies to perform certain actions (such as moving towards a known enemy location). The latter are specified for a variety of different modes (for example, an entity may have different behaviour when they have been injured).

Lauren and Stephen motivate MANA's simple model by noting that apparently more detailed and realistic simulators (they name CAEn and Janus as examples of these) don't necessarily produce more accurate results, particularly with regard to the 'fat tail' of low-probability outcomes. They note that existing high-fidelity simulations often overemphasise the importance

of physics and material behaviour over other (equally important) aspects such as situational awareness, sensor effectiveness and command and control difficulties.

In [58], Galligan discusses an extension of MANA that adds explicit modelling of entity situational awareness (SA). He then uses this to explore the factors that contribute to friendly fire casualty rates in a NCW system. This was a fairly simple study, looking at the effects of bandwidth and latency on the number of friendly-fire casualties inflicted in a single scenario using one sensor and one shooter.

The MANA approach is promising, in that (particularly with the more detail SA representation described by Galligan) it can quite directly represent SoS by providing an automata for each agent and revealing the behaviour that emerges. However, the MANA models are very simple. Lauren and Stephen note that MANA avoided support for arbitrary behaviour rules because this generally lead to *"modelling platforms that resemble high-level programming languages themselves"*.

The limited expressive power of the MANA framework means that it cannot be used to directly express specific real-world SoS. For example, a proposed SoS description might well include description of coordination protocols between two entities (such as how they would communicate to synchronise their actions in a combined air-and-land attack). This behaviour could be expressed, for example, as a UML sequence diagram, but it would be difficult or impossible to render that interaction protocol in a MANA model.

None of the extant papers on MANA discusses how one would go about converting a paper description of a SoS into an adequately representative model for any given analysis. The models and results from the Galligan study are abstract — they don't apply directly to any real system. The paper does not discuss how such validation might be performed.

2.5.5 Using SEAS for Collateral Damage Modelling

In [59], Brooks et al. describe an approach that uses multi-agent simulation to model collateral damage (friendly and neutral casualties and destruction of buildings) caused by military actions. This is similar to the work using MANA described in Section 2.5.4, in that the component systems within the SoS are represented by individual agents who move around within a space and a perform combat actions according to pre-specified behaviour rules.

The approach is similar to that used by the QHI system for automated HAZOP (see Section 2.5.1), in that a model (of a system and scenario) in the format used by the SEAS simulation engine is input to a pre-processor which manipulates the model based on a set of parameter values (for example, parameters include the number and density of neutral buildings and civilians). The simulation is then run and a number of collateral damage metrics are computed.

The SEAS engine appears to be more expressive than the MANA model, in that it allows arbitrary rule sets to be used to describe agent behaviour. As with the published MANA work, however, there is no discussion of the modelling approach that was used to derive adequate sys-

tem and environment models in the first place, and there is no particular method described for using the available parameters to systematically explore the space of possibilities. No particular consideration is given to the validation of the models used.

2.5.6 Multi-agent Simulation for Hospital Evacuation

Johnson, in [60], describes the use of multi-agent simulation to evaluate the effectiveness of hospital evacuation procedures. Although the applicability of this to the military and transport SoS that are the concern of this thesis is not direct, there are a number of aspects that make it relevant.

The approach uses models of patients, staff and the interactions between them to study how long it would take to conduct a complete evacuation of a particular modelled hospital. The patterns of behaviour exhibited by each agent type can be based on real-world duties, procedures and measurements. Movement rates, for example, can be derived from studies of the speed at which the participants move under real-world experimental situations, such as pushing a wheelchair (with occupant) of average weight. When this information is incorporated into the multi-agent simulation, the emergent effect of the rules and values can be observed.

Similarly, layouts of actual buildings can be derived from CAD (Computer-Aided Design) files created by the original architects, so as to ensure that the layout fits closely to the real hospital. Again, in simulation the emergent behaviours of multiple agents are combined with their environment to produce plausible emergent results.

Once the whole model is in place, parameter values (such as the number of patients in particular mobility categories) can be varied and the results observed. As with the friendly fire analyses described above, the primary outputs are a real-time visualisation of the running simulation and measures of the relationship of parameter values to mean evacuation times (the latter using Monte Carlo techniques).

The use of visualisation (which is extremely common in multi-agent simulation because most multi-agent models have an obvious visual representation) allows some aspects of how undesirable situations arise to be observed. For example, an observer might be able to see that a large number of beds or wheelchairs have blocked a major thoroughfare. This provides some knowledge of the emergent 'mechanisms' of the model, however it is difficult to claim that all such mechanisms have been discovered without requiring that analysts observe (and, more importantly, *understand*) all simulation runs. Visualisation can be a useful tool in helping stakeholders understand a complex simulation. However, to rely upon it entirely as a means of explanation raises questions of completeness.

As well as validating the agent model performance by reference to small, real-world observations or experiment, the resulting evacuation times can be validated by performing real-world evacuations. Johnson notes, however, that performing such evacuations is practically and ethically difficult. Analogous real-world evaluation is at least comparably difficult (and often more so) for the large-scale SoS with which this thesis is concerned. Furthermore, evaluation of sta-

tistical outputs is, of itself, of limited value — hazard analysis requires that the mechanisms by which those statistics arose be discovered and described. There is a clear need for other ways of gaining confidence in the validity of analysis results.

2.5.7 SpecTRM-RL

Leveson, in [40], describes how the SpecTRM-RL tool can support STPA analysis as described in Section 2.4.2.5. The tool allows a specification of the system under analysis to be represented formally, and for automated analyses to be performed on it. STAMP processes can be represented in the tool using a combination of (continuous) process variables and discrete logic.

Leveson notes in [61] that a developed SpecTRM-RL model can being linked into a simulation model to explore how the model behaves in a more explicit context, but it is unclear from the published work what the practicality of this is in the general case.

SpecTRM-RL requires its own, STAMP-specific models be developed before automated STPA analysis can be performed, and there is no published process for doing this. It can also be noted that Leveson promotes the STAMP model partly on the basis that it can model the system context well above the technical level (taking into account the operating organisation and even governmental bodies with which they interact). It is not clear, however, how SpecTRM-RL can allow these aspects of the model to be considered during automated STPA, as its modelling abstractions are unlikely to adequately represent such complex entities.

The use of SpecTRM-RL for automated STPA analysis is potentially a strong approach to SoS hazard analysis problems, but the descriptions in published work are currently lacking. It can be observed that many of their examples (for example the TCAS collision avoidance system) are based on systems that are already known to have safety-critical problems (as illustrated by the Überlingen air crash [62]).

2.5.8 Conclusions on Automated Hazard Analysis

Reviewing the work on automated hazard analysis that might be relevant to SoS, several conclusions can be drawn. The Automated HAZOP work is strong on several measures (it explores the space of possible hazards (within the constraints of the model provided), it allows causal explanations to be easily found, and there exist mature implementations). However, the models it works over cannot directly be mapped to our concerns for SoS safety.

Multi-agent simulation approaches (advanced by Lauren and Stephen [57], Brooks et al. [59], and Johnson [60]) are potentially highly applicable to SoS because a clear mapping can be established between SoS component systems and agents that represent them. However, the safety-related multi-agent approaches are marred by their lack of explicit modelling processes, the difficulty in discovering the mechanism by which accidents come to happen, and the lack of attention to validation of the models.

Leveson's work on automated STPA is interesting but difficult to evaluate or apply given the

currently published work. The lack of explicit modelling and analysis processes is particularly significant.

2.6 SoS Modelling and Analysis

In the conclusions of the previous section, it was observed that the automated hazard analysis techniques that are applicable to SoS are generally lacking in terms of modelling process. There has been considerable attention in the literature to SoS modelling in the general case, and relevant work is reviewed in this section.

2.6.1 Object-Oriented Modelling of SoS

Caffall and Michael [3] note that *"Our tools for reasoning about a system-of-systems typically consist of little more than a 'sticks and circles' diagram"*, whereby they are referring to a diagram in which each component system is a circle, and all means of communication are represented by simple lines or arrows. They suggest that the overuse of such trivial notations has lead to SoS that are tightly coupled and more complex than necessary. They propose an object-oriented modelling approach for SoS, using a notation not dissimilar to the UML class diagram. They use the United States Ballistic Missile Defense System as an example.

In their approach, each element in the system is treated as an object, and a class diagram is drawn up to show the class (not instance) relationships. As with UML class diagrams, cardinality can be shown. Each class has methods that describe its external functionality, i.e. the services and operations that other classes in the system can use. An example of a SoS represented in their notation is shown in Figure 2.3.

They propose using inheritance to solve the problem of heterogenous systems, and to an extent the problems of dynamic reconfiguration as well. Abstract classes provide models of generic component systems, for example 'weapon' and 'projectile', with the methods covering the generic functionality. Subclasses of those then develop more specific variations, such as 'ground-based missile launcher', and further subclasses can detail sub-categories or specific models.

The actual purpose of Caffall and Michael's notation is unclear. The descriptions are provided at a very high level, and although basic functions and relationships are identified there is little that can directly be inferred about the actual behaviour of the system. The notation has no semantics, and no way to describe the sequences of actions that are expected to occur (at a level below, for example, general relationships of the form 'controls' or 'releases' as shown in Figure 2.3).

It can also be noted that the notation does not distinguish between instances of the same component system type — as in standard UML class diagrams, the notation describes the relationships between types of system, not between individual instances of those types.

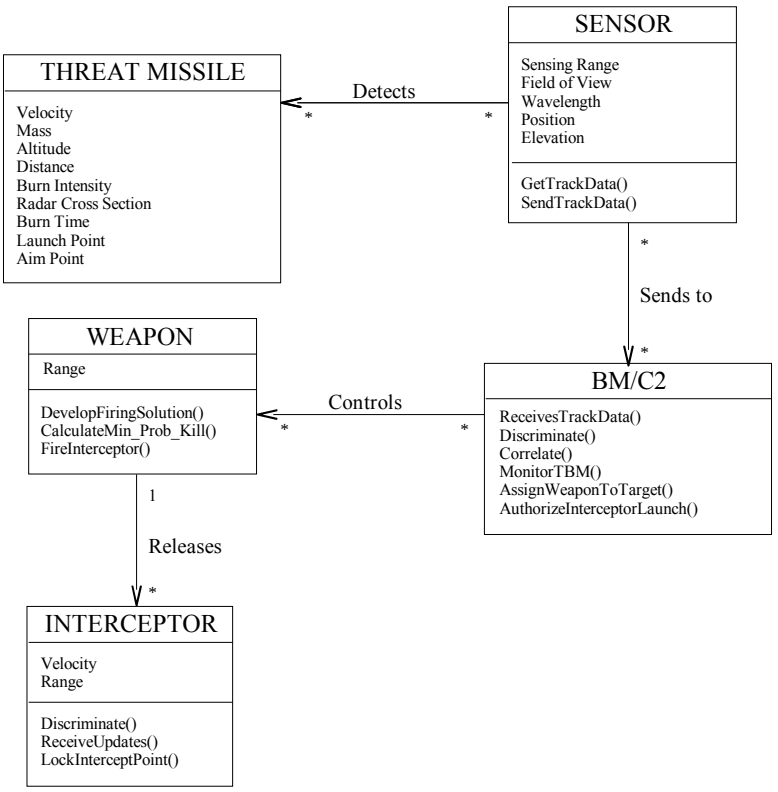

Figure 2.3: Class Diagram of Hypothetical Missile Defence System-of-Systems (from [3])

2.6.2 DoDAF and MODAF Architectural Models

DoDAF (Department of Defense Architecture Framework) is an architecture description framework developed by the US Department of Defense. Volume I of the DoDAF description [63] states that DoDAF *"defines a common approach for DoD architecture description development, presentation, and integration for both warfighting operations and business operations and processes."* MODAF is a DoDAF variant developed by the UK Ministry of Defense, which adds some additional viewpoints and artefacts ('products' in MODAF/DoDAF terminology) to support MOD concerns.

In [64], the MOD notes that *"Modern warfare is fast changing and the systems that technology is now making available are in themselves faster, more complex and more adaptable than ever before. The combination and orchestration of these systems in concert with operational planning introduces a level of complexity never before experienced"*. They propose the use of MODAF as a way to manage this complexity by allowing stakeholders to view the system from a variety of viewpoints and at an appropriate level of abstraction.

MODAF and DoDAF allow a SoS to be described from several 'views': the Operational View (OV), which describes component systems and behaviours of the SoS at a high level, the Sys-

Product	Name	Description
OV-1	High-Level Operational Concept	Describes a mission or scenario, in terms of the operational elements (component systems) and the context. Consists of OV-1a (an illustrative graphic) and OV-1b (text).
OV-2	Operational Node Relationships Description	Shows the operational elements in the system and the information flows between them. The latter are represented in the form of 'needlines' that describe the information needs that hold between each node.
OV-5	Operational Activity Model	Describes the activities that the system will carry out, and how those activities are related in terms of flows of orders and information.
OV-6a	Operational Rules Model	Contains rules describing how the SoS behaves under various circumstances and conditions.
OV-6b	Operational State Transition Description	Expresses the overall behaviour of the SoS as a (optionally nested) state machine.
OV-6c	Operational Event-Trace Description	Shows how the operational nodes communicate and interact in a particular scenario.
OV-7	Information Model	Describes the structure of the information exchanged between the nodes in the system.
SV-4	Functionality Description	Documents the necessary functionality that must be provided by the SoS in order to carry out its objectives.
SV-5	Function to Operational Activity Traceability Matrix	Relates the functionality of the overall SoS that was identified in SV-4 to the nodes within the system.

Table 2.5: MODAF Products

tems Views (SV), which contains more detailed information about the computer and communications systems that are used by the SoS, and the Technical View (TV), which describes relevant technical standards. The All View (AV) contains the highest-level view of the system along with other information that is relevant in multiple views.

MODAF extends these with the addition of the Strategic View (StV), which identifies future development needs for the SoS, and the Acquisition View (AcV), which describes the steps that are being taken to meet those needs through system acquisition. With the exception of these views (which will not be explored further here), references to DoDAF and MODAF in this thesis are largely interchangeable.

For each view, a number of individual 'products' are defined. Between the five views there are more than thirty such products. Examples of these (the most important for this thesis) are described in Table 2.5. Examples of most of these can be found in the MODAF model presented in Appendix A.

A full MODAF model can represent considerable detail about a SoS. In this, it is much stronger than Caffall and Michael's notation (see Section 2.6.1). Its operational nodes correspond to the

entities in Caffall and Michael (although it is noted by Mittal in [65] that DoDAF doesn't enforce entirely consistent use of entities between different products in the OV), but it adds detailed information about entity behaviour (in OV-5 and OV-6c) and some detail of the technologies used (in the SV products). It also has the advantage of being mandated for current and future MOD SoS development.

It is not clear, however, how hazard analysis could be performed over a MODAF model. Some partial analysis opportunities are apparent, for example a HAZOP-style analysis could be performed on the information flows in OV-2 or the command and control relationships in OV-4. There is no currently extant general approach, however, which integrates multiple products and exploits the combined information.

Although some official DoDAF publications (e.g. [66]) discuss the potential for 'executable architectures' based on DoDAF, standard DoDAF models do not have a formal or executable semantics. Mittal, in [65], observes that DoDAF has most of the information needed to build executable simulation models, but this is not directly translatable (without application of significant engineering skills and effort). Indeed, Mittal states that *"DoDAF has completely overlooked* [modelling and simulation techniques] *as a possible means to evaluate design, capabilities, and planned expansion of current architectures. There is no provision for testing the constructed system, either in OV or in SV."*.

Given this lack of strong semantics, questions can be raised about the likely quality and comprehensiveness of MODAF models in practice. It is plausible that many MODAF models that are produced will be inaccurate or self-inconsistent in ways that are acceptable for manual use but which hinder the creation of sensible machine-analysable models. This concern is supported by Zinn, who explains in [67] that despite working and studying within the US Air Force (which is committed to the use of architectural models) he had great difficulty finding adequately complete models. For example, he had to rule out one candidate model because it had an OV-5 but none of the OV-6 products, making it impractical to produce simulations from it.

Overall, the combination of detailed architectural description with mandated DoD and MOD implies that MODAF has potential as a model for SoS hazard analysis. There is a need, however, to develop an analysis technique that takes full advantage of the model by exploiting the combined information from multiple products.

2.6.3 Cellular Automata Models

Ilachinksi, in [68], discusses the use of cellular automata models for the prediction and understanding of military performance, and presents the EINSTein[1] simulation engine as an example of such. The MANA simulation system, discussed in section 2.5.4, is another example of such a model.

[1] 'Enhanced ISAAC Neural Simulation Toolkit'. EINSTein's predecessor was ISAAC, 'Irreducible Semi-Autonomous Adaptive Combat'

Ilachinski promotes the use of such *'bottom-up, synthesist'* models over older 'reductionist' models (such as those derived from the Lanchester Equations originally presented by Lanchester in [69]) on the grounds that they better represent the dependency of combat situations on complex, non-linear interactions, temporal and spatial factors, and human behaviour patterns. They allow the behaviour rules of the individual automata to lead to the emergence of self-organisation and, potentially, effective military tactics.

Attempts have been made to apply such models to safety analysis, but those apparent from the literature are unsophisticated (see, for example, the discussion of MANA in Section 2.5.4).

It can be noted that much of the effort in cellular automata models is explicitly directed towards discovering general principles of military operation and complex, self-organising systems, rather than analysing particular real-world systems and situations. For example, Ilachinski states *"Fundamentally, EINSTein addresses the basic question: 'To what extent is land combat a self-organized emergent phenomenon?' Or, more precisely, 'What are the conditions under which high-level patterns (such as penetration, flanking maneuvers, attack, etc.) emerge from a given set of low-lying dynamical primitive actions (move forward, move backward, approach/retreat-from enemy, etc).' "* [68]

Perhaps as a consequence of this, the extant cellular automata models lack expressive power — although they can demonstrate some impressively realistic actions in the general sense (Ilachinski describes how simple optimisation using genetic algorithms can produce a variety of real-world tactics in EINSTein such as a 'hammer and anvil' manoeuvre), they cannot be used to represent specific real-world platforms and doctrines directly because they do not allow the user to specify arbitrary procedures or cognitive models to guide the behaviour of the entities in the simulation.

It can also be observed that, although EINSTein and MANA provide statistics and visualisation for discovering the system-level behaviour that emerges from a particular set of parameters, neither has any particular tools for discovering precisely *how* the interactions of the automata lead to a particular observable behaviour. This deficiency is common in agent-based simulation systems (see, for example, the discussion of the hospital evacuation models in Section 2.5.6). This is unfortunate as the discovery of the 'how' is a crucial part of hazard analysis.

2.6.4 Derivation of DEVS Simulations from DoDAF

Mittal, in [65], proposes a method for deriving DEVS (Discrete Event System Specification) simulations from DoDAF models. As was noted in Section 2.6.2, Mittal has criticised DoDAF for its lack of explicit support for simulation. He specifies a method for supplementing DoDAF models with additional information and then using the augmented model to derive a DEVS-compliant executable simulation (as described by Zeigler in [70]).

Mittal criticises DoDAF for its lack of detail on the interfaces between the operational activities expressed in product OV-5, the lack of a fixed set of model entities across all products (which would correspond to the (necessarily) fixed set of entities in a simulation model), and the total

disregard of operational performance or timing issues. He proposes that DoDAF be extended with additional artefacts OV-8 ('activity component description'), which maps quite directly to a DEVS model, and OV-9 ('activity interface specifications'), which supports the testing of the DEVS model.

Implementing Mittal's method would mean changing the way that DoDAF models were developed, principally because Mittal specifies rules for developing the existing products (for example, he requires that a precise activity-entity mapping be developed before product OV-6b and after OV-5). It is unclear how this is supposed to apply to existing DoDAF models.

The work presented by Mittal does not directly address safety issues. This is compounded by its lack of attention to spatial representation — it seems primarily concerned with describing the conceptual structure of the modelled system, rather than with its embodiment in real-world spatial agents. It appears to support an fairly abstract model that is primarily of use for verifying simple functional and performance characteristics, but it can be questioned whether it is applicable to the kinds of emergent safety properties with which this thesis is concerned.

The approach emphasises the use of software testing techniques, and related concepts of coverage, in order to evaluate the effectiveness of the model (although Mittal gives few details). In the published work, little consideration is given to explicit scenarios and external SoS environments. In general, the analysis that Mittal proposes the DoDAF-derived simulations be subjected to is not developed in detail.

Overall, Mittal's approach is promising in that it starts from DoDAF models (which are currently often available and are likely to become widely so) and combines the information in the various products to develop an executable simulation model that can then be analysed automatically. The precise form of model developed, however, does not directly support the kind of analyses with which this thesis is concerned.

2.6.5 Generation of Multi-Agent Models from DoDAF

Zinn, in [67], presents an initial investigation into the use of DoDAF models to develop multi-agent simulations. This is similar to Mittal's work discussed above, except that it uses the Systems Effectiveness Analysis Simulation (SEAS) multi-agent simulation engine rather than the DEVS formalism.

The SEAS engine is broadly similar to the EINSTein and MANA models discussed in Section 2.6.3, in that it models forces as discrete entities acting autonomously within a simulated environment, but it avoids some of their limitations by allowing the expression of arbitrary rules and procedures using a domain-specific programming language.

Zinn notes that the production of multi-agent models can involve considerable effort, and proposes an approach where pseudocode for agent behaviour descriptions is generated from OV-5 and OV-6a. The resulting pseudocode can be readily translated into the Tactical Programming Language (TPL) format used by SEAS by any programmer familiar with the latter, without a

particular need to understand DoDAF.

Similarly, details about platform performance criteria can be derived from SV-7 (although Zinn notes that in practice the detail provided is only at the subsystem level), and information on communications links, message types and characteristics (such as bandwidth and reliability) is taken from OV-3 and SV-2 (although, again, Zinn noted that in the DoDAF models he had access to such detail was generally missing).

No particular analysis is proposed in Zinn's work — he gives an example of using the model for military effectiveness analysis, producing statistical output.

Zinn's modelling approach doesn't consider safety analysis. Although the modelling of temporal and spatial properties provided by SEAS does provide strong grounds for representing accidents within the simulation, no method is provided for deriving explanations as to why such events happened. Nor is any particular consideration given to the variability of agent properties or behaviour, which might provide grounds for deriving accident outcomes from otherwise safe scenarios.

Although SEAS does provide facilities for modelling the environment of the SoS, including hostile forces, weather and terrain, Zinn gives no particular attention to the derivation of appropriate scenarios and environmental conditions that need to be considered for a particular analysis. The environment used by the example in Zinn's thesis is arbitrary, being chosen merely because Zinn adapted his example scenario from an existing SEAS-format model.

As noted above, Zinn provides a partly-automated approach to the translation of the OV-5 and OV-6a products into pseudocode, but this relies on those products being presented in the formats prescribed by Telelogic's System Architect (IDEF0 and IDEF3 for OV-5 and OV-6a respectively). This use of products is not universal, and indeed the published DoDAF and MODAF technical specifications ([66, 71]) propose the use of use of other notations (UML activity diagrams for OV-5, free text for OV-6a). Zinn's work may not, therefore, be applicable to many of the DoDAF models that are currently being produced.

From the description in Zinn's work, and in [59], the SEAS simulation package appears to have strong potential for SoS modelling, as it has the flexibility to model specific individual systems and procedures. It is not, however, generally available outside of the US military and their sponsored projects.

2.6.6 Conclusions on SoS Modelling

A range of modelling approaches have been applied to SoS, and some of these are promising. DoDAF/MODAF provides a standardised way for a variety of stakeholders to express proposed SoS architectures and communicate about them, although it is difficult to analyse in its current form. Cellular automata models can express many of the important properties of complex systems, but are limited in their expressive power and the ease with which they can be analysed. The work by Mittal and Zinn provides a systematic way to derive executable models

from DoDAF models, but the models so derived are not particularly amenable to safety-related analysis.

None of the approaches described here have been adapted for systematic hazard analysis of SoS.

2.7 Conclusions

Although there has been some expression of concern in the literature over the safety challenges presented by SoS, there is little published work that attempts to tackle the core difficulties (the primary exception being that of Leveson, in [40]).

There are a huge range of available manual techniques for hazard analysis, but the majority of traditional techniques scale poorly to the decentralised, dynamic structures and complex sets of multiple goals that characterise SoS (discussed further in Section 3.1.1).

There are a variety of automated hazard analysis techniques described in the literature, but many are not directly applicable to SoS, or lack systematic means for exploring the space of possible failures and deviations. For those techniques that are potentially SoS-capable, the credibility of the models used is often in doubt, principally because the modelling process used is typically poorly documented.

Beyond safety, there are various ways to model SoS, both on paper and in the form of simulations. Current interest in systems modelling (motivated by the increasing complexity of deployed systems) has lead to both the US Department of Defense and the UK Ministry of Defence mandating standards for (paper) models of SoS which can potentially be exploited in SoS hazard analysis. There are a variety of promising approaches to building SoS simulation models, although those currently extant are generally deficient in terms of either (a) the lack of an explicit modelling method and a lack of expressive power or (b) the lack of attention to aspects such as spatial modelling that are critical for safety.

Taking the work described above, and considering the identified shortcomings, the next chapter begins to develop an approach to SoS hazard analysis.

Chapter 3

Establishing Hazard Analysis as Part of SoS Development

This chapter shows how hazard analysis can be performed in the context of a SoS development process. Building on the literature review in Chapter 2, it presents the outline of a solution to the SoS hazard analysis problem. This provides a framework; the most critical and difficult parts of the process are developed further in Chapters 4 and 5.

3.1 Concepts

3.1.1 Defining Characteristics of SoS

Drawing on the definitions of 'SoS' reviewed in Section 2.1.1, it was observed by the author and Hall-May in [72] that SoS share a set of common characteristics, the combination of which marks them as distinct from mere 'systems'. It is the consequences of these characteristics that raise concerns for SoS safety. Some of these characteristics are definitional, in that anything that does not exhibit them cannot reasonably be considered a SoS, while some are merely common. The case studies used in this thesis exhibit most or all of the characteristics.

The characteristics identified in [72] as *definitional* are:

SoS has overall goals All the entities in the system contribute to achieving some set of overall system goals. This is definitional in that in the absence of such shared goal there is no SoS, just a group of loosely-associated systems.

SoS contains multiple component systems The SoS is made up of multiple components which are distinct systems in their own right, and have a lifecycle outside of the SoS.

Components have local goals In addition to sharing the overall goals of the SoS, each component system has its own distinct goals. Typically, these goals will include self-preservation and the maintenance of an adequate degree of safety, along with sub-goals that represent the component's contribution to achieving overall system goals. For example, a UAV may assume responsibility for surveying part of the area that the overall SoS intends to survey. It is not meaningful to consider a system to be an SoS unless the components have goals that are distinct from those of the overall SoS.

It should be noted that local goals may come from the component itself (e.g. self-preservation) or be handed down from a higher authority (e.g. a target that is to be attacked).

Components have some level of autonomy The components are able to respond to external stimuli, to changes in their environment, without instructions from outside. Components can vary in their levels of autonomy; some will have only the ability to find routes between waypoints, while others can plot their own waypoints as part of achieving mission goals. Other components may be able to formulate their own goals based on general instructions, while others still will have no autonomy at all. A set of distinct levels of autonomy is developed by Clough in [73].

Components need to collaborate The relationship of the individual capabilities of the components, when compared to the scale of the SoS's overall goals, is such that components need to collaborate in order to achieve them. It may also be that component systems have to cooperate to share a scarce resource. Examples include the need to mass forces to attack a fortified enemy position, or for aircraft to share airspace safely.

Components communicate with each other There is some form of information exchange between components, such as by radio or a computer network.

The characteristics that are not definitional, but are *common*, are:

Components have individual capabilities The individual components of the SoS have varied capabilities, and the overall capabilities of the SoS are derived from these. No single distinct component can be built or acquired that has the capability to accomplish all the SoS goals. All components have some lack of capability, either with respect to types of activity or geographic area covered.

This characteristic is very clear in the military case, where vehicles and groups are specialised for particular purposes — a bomber aircraft cannot hold a town, and a dismounted infantry section cannot provide reconnaissance over a hundred-mile radius.

Components are geographically distributed The component systems are not part of one physical whole; they may be distributed over a wide area such as a building or a country.

Components are physically mobile The components can move with respect to each other under their own power, and this is a normal part of routine operation. I.e. many components of a SoS are themselves complete vehicles.

Components communicate using ad-hoc networks Rather than having fixed lines of communication (such as point-to-point wires) or pre-arranged radio channels, communications links are often arranged between components as needed.

Components are heterogenous SoS are not, in general, designed in a single effort — they are assembled from diverse existing systems, many of which will have previously been fulfilling other roles in other SoS. The lifecycle of the component systems may also be much shorter than the lifecycle of the SoS itself, leading to a change in the components of the system over time.

The result is that components may have been designed separately, by different groups or organisations at different times. When each component was designed, the set of peer components was not known, nor was the configuration of the SoS as a whole.

For example, a military unit may field several different kinds of armoured vehicles for different purposes. Each may have been built at a different time, by a different organisation, for a different enemy and different doctrine. Similarly, the aircraft in a controlled airspace system will be diverse in terms of manufacturer and in terms of the aerospace environment (such as traffic levels) and air traffic policies that were in force at the time of their design.

3.1.2 Contribution of SoS Characteristics to Hazards

The characteristics of SoS identified in the previous section are interesting because they can contribute to SoS hazards and hence to accidents. Table 3.1 discusses some of the ways in which each characteristic can bring about a hazard.

The table above can be illustrated by reference to historical SoS accidents. For example, the existence of overall system goals commonly focuses the activity of multiple entities on the same location at the same time. This can be seen in the Lake Constance mid-air collision [62], where the system goal of organising air traffic into well-defined flight paths made it plausible for two aircraft to be moving at exactly the same altitude. There are also a host of military accidents stemming from plans of the form “infantry advance behind artillery” where the location that was to receive artillery bombardment was the same location that needed to be occupied on the ground (see House [32] for some discussion of this).

The conflict between local and system goals can be illustrated by the Black Hawk friendly fire accident (see Section 1.1.1), where the system-level goal “maintain safety by enforcing

Characteristic	Possible Contribution to Hazards
System goals	System goals focus multiple entities on the same place at the same time
Multiple components	In-system fratricide, collision etc requires multiple components
	Multiplicity of components increases complexity of system (both for analysis and for control in the field)
	Loss of component breaks system mechanism or functional capability
	Too many components for some resource (e.g. bandwidth)
	Too few components to achieve a certain function
	Component inappropriately substituted
Component situational awareness / worldview	Two components disagree on world state / get out of sync
	One component lacks information about others (particularly combat ID)
Component goals	Conflict between local and system goals
	Conflict between local goals of two components
Component capability	A component's capability doesn't match its assigned role
	Component doesn't have expected capability
	Capability bounded or limited in some way (range, capacity per unit time, workload etc)
Geographical distribution	Component only has local knowledge
	Two entities separated by terrain (so e.g. communications problems)
Mobile systems	Moves out of visual or communications contact
	Moves into trouble (terrain, another entity)
	Component not located where it is expected to be
Autonomy	Component acts out of sequence, or at an inappropriate time
Collaboration	Collaboration sequence does not occur as planned
	Conflicting orders/instructions
Comms	Loss of messages, or ability to communicate
	Widespread loss of comms (e.g. through jamming or infrastructure damage)
Ad-hoc comms	Local communications overload (e.g bandwidth in area)
	Unexpected emergent network vulnerability (e.g. one component forms vital node)
Heterogeneity	Technology mismatched between components
	Two different components fulfil the same role, but in different ways

Table 3.1: Contributions of SoS Characteristics to Hazards

strict rules of engagement" was compromised by the local goal of the F15 pilots "maximise air combat effectiveness".

The effects of limited component capability can be seen in the USS Vincennes accident (see Section 1.1.2), where the radar and AEGIS operators on the Vincennes were unable to adequately track and comprehend the aircraft around them under combat conditions.

3.1.3 The Role of Humans in SoS

Traditionally, safety engineers have worked with the assumption that human operators were outside the system. In mechanical and electrical systems with a clear boundary (a category

which can include even such advanced systems as modern aircraft and autonomous robots). Their modelling could therefore concentrate on the electrical and mechanical components (and more recently, software components) that comprised the system. This emphasis can be seen in the common tendency to blame accidents on 'operator error' (see Johnson and Holloway in [74] for some discussion of this issue).

As noted in Section 1.2.2, the overall behaviour of a SoS emerges from the behaviour of its components, and therefore on the behaviour of the humans that are themselves components (e.g. in the case of the soldiers on the ground) or that are operating those components (e.g. in the case of aircraft pilots). SoS analysis, therefore, needs to be carried out at a level that *includes* the human operators.

Prediction of likely human behaviour is extremely difficult, but it is widely recognised that it is necessary. For example, the work of Leveson (see Section 2.3.3) treats humans as being part of the system at a variety of levels, as does much of the recent work derived from military modelling and simulation (see, for example, Sections 2.5.4 and 2.5.5).

3.1.4 Ownership and Control of SoS

One concern with SoS is who, if anyone, 'owns' or controls the whole system. It can be seen that answering this is potentially difficult, especially when the SoS results from combining systems (and, indeed, smaller SoSs) that have never previously been combined. Military SoS are particularly problematic because of the frequency of multinational coalition operations where no clear overall command can be established. There is also the problem of 'reachback', whereby modern network technology allows a SoS deployed in the field to incorporate resources owned and operated by other agencies located in the SoS's country of origin. These assets may be vital for SoS operation but under a wholly separate chain of command.

Nevertheless, if a SoS engineering effort is to proceed, someone (or at least some organisation) must take responsibility for the overall engineering and management of the SoS. Fundamentally, they must take responsibility for the *integration* of the component system that form the SoS. In practice, this is likely to involve them in defining the SoS architecture and deciding on its operational policy.

The *SoS owner* is responsible for the behaviour of the overall integrated SoS, in ensuring that meets its capability requirements. Primarily, this means that they manage the context in which the various component systems operate, but they must become aware of, and resolve, the effects of interactions (intended or otherwise) between the component systems. In the UK Ministry of Defence there is the concept of a 'Senior Responsible Owner' who fulfils this role of SoS owner during operation, although it is harder to identify a single owner responsible for longer term SoS development [75].

The *component system owners* (CSOs), by comparison, are responsible for the identified component systems that make up the SoS — each component system has a defined owner. Pyster and Gardner, in [1], refer to these as "prime contractors" although many relationships with

the SoS owner are possible. It likely that such component system owners will be drawn from multiple organisations.

The CSO does not need to be the manufacturer, although where such an organisation can be identified there will be a need for the CSO to liaise with them. Rather, as was the case with the SoS itself, the CSO is an identified body that *takes responsibility* for the development and behaviour of their specified component system. In many cases, the CSO will have performed integration of multiple manufactured artefacts, or the combination of an artefact with human assets (as in the case of, for example, a fighter aircraft, which is procured from a supplier and then combined with a human pilot who has received particular training).

The critical role of the CSO with respect to the SoS is to analyse, report on and control the system's response to the SoS context. To perform this, they will need to stay 'in the loop' with the SoS owner, responding continually to changes in the SoS context (which will include changes caused by the decisions of other SOs).

In the paragraphs above, 'development' is used as a shorthand; in reality, SoS development is likely to involve large amounts of procurement of commercially-available assets or adaptation and reuse of existing assets.

In SoS development, despite the potential political difficulties, it is critical that everything is owned by someone. The SoS itself must have an owner, and every identified component system must have an owner. The rest of this thesis assumes that, in all cases, the owners of the SoS and component systems are known.

3.2 SoS Safety Lifecycle

By combining the conventional system safety 'V' model presented by Pumfrey (see Section 2.4.1) with the SoS engineering lifecycle presented by Pyster and Gardener (see Section 2.2.1), a SoS safety lifecycle can be derived. This uses the overall structure provided by Pyster and Gardener's double-V model, but uses the specific activities from the safety lifecycle. The lifecycle is shown in Figure 3.1.

The inner 'component system V' cannot be seen clearly in the diagram, but it is unchanged from the lifecycle presented in Section 2.4.1. It can be observed, however, that the SoS V is slightly different in structure to the component system V: the first safety activity has become Accident Identification, and Hazard Identification and Risk Assessment have been pushed later into the lifecycle, to coincide with architectural design. This change is necessary given the complexity of SoS and their intractability to conventional manual analysis; the behaviour of a SoS cannot be known in any detail until the architecture is described.

The work presented in this thesis is primarily concerned with SoS-level Hazard Identification (SoS-HI) and Risk Assessment (SoS-RA) in support of efforts to perform SoS-PSSA based on a SoS architecture. Some notes on the application of the work to other activities and lifecycle phases will, however, be given at various points in the text.

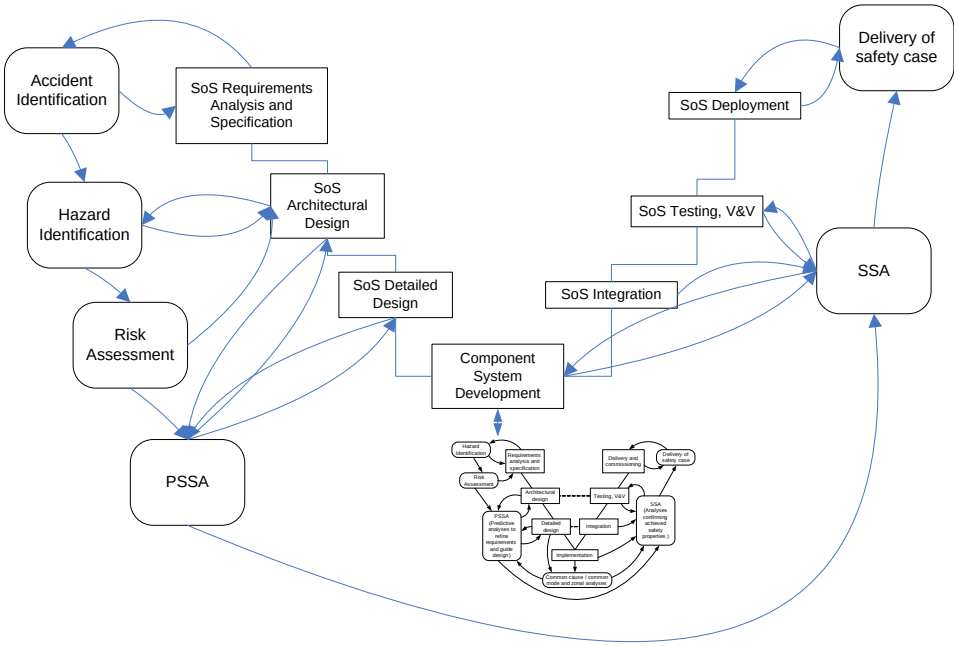

Figure 3.1: SoS Safety Lifecycle Diagram

3.3 SoS Hazards

It is obvious that accidents can occur in SoS. Any SoS that contains vehicles or weapons systems can cause serious damage to humans or to valuable material goods. Prior to the occurrence of such an accident, there will be a point at which the component system in question is exhibiting a hazard. Some of these hazards will be straightforward because of changes in the state of the component system; an aircraft may crash because one of its engines fails.

The work in this thesis is not, however, concerned with such simple hazards that have their apparent genesis wholly within a single component system. Such hazards are amenable to conventional hazard analysis techniques. Rather, it is concerned with component system-level hazards that occur *because of the state of the SoS*; in other words, because of a SoS-level hazard.

A 'SoS accident' involves a *stimulus* of some kind coming either from the environment (such as the appearance of a third-party entity that is not part of the SoS) or from within the system (such as an error within a component system). This stimulus leads to a change in the state of one component system (*component system state*), which in turn causes changes of the state of one or more other component systems (*SoS state*). Eventually, one or more of the component systems reaches a hazard state (a *component system hazard*), which leads to an accident.

A SoS hazard is extant whenever there is a component system within the SoS that can cause another component system to enter a component system hazard state without any further abnormal behaviour. A SoS hazard is also extant if there are *multiple* component systems that, by interacting, can cause a component system (either one of them or a third party) to exhibit a

component system hazard, again without further abnormal behaviour.

It is worth noting that the terminology of 'failure' is potentially misleading when considering SoS. A component system may be operating entirely consistently and correctly with respect to its own specification *and* intent and yet there may still be a 'failure' with respect to the overall intent of the SoS.

SoS hazard A condition of a SoS configuration, physical or otherwise, that can lead to an accident.

SoS accident An accident that occurs because of a SoS hazard.

SoS hazard analysis, the topic of this thesis, therefore involves finding out what SoS hazards can be exhibited by the SoS, and how they arise from stimuli at the level of the individual entities. This is illustrated by the 'double bow-tie diagram' in figure 3.2 (the figure is derived from one presented by Pumfrey in [2] for non-SoS hazards). In the diagram, several states or events at the component system level occur together and cause a SoS hazard, which leads (in the presence of a contributing factor) to a component system hazard. At this point, certain other contributing factors may cause an accident to occur, while others may cause the system to transition back into a safe state. A complete SoS hazard analysis must capture all the paths by which component system states could lead to SoS hazards, by which SoS hazards could lead to component system hazards, and by which component system hazards could lead to accidents.

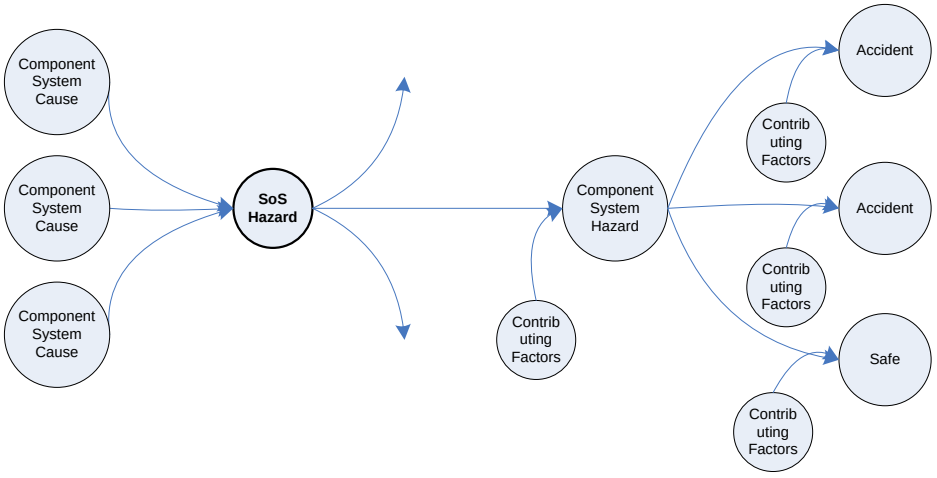

Figure 3.2: The Development of Stimuli Into SoS Hazards and Accidents

The set of actual component system-level accident types that are possible for a given SoS is actually very small, being limited to the product of the component system types with basic types of environmental features. A basic list of types, covering the civil aviation and military domains, would be:

- SoS component system collides with SoS component system

- SoS component system destroys SoS component system with weapons
- Sos component system collides with external entity
- SoS component system destroys external entity with weapons
- SoS component system causes wider environmental damage (e.g. by releasing the toxic waste it is carrying)

Component systems within a SoS interact over channels. Obvious examples of such are radio networks and wired computer networks; a less obvious example is visual gestures. Even less obviously, there is the case of interaction through 'unintentional gestures' whereby one component systems sees another component system take an action, infers the other component system's intentions from that action, and takes action itself in response.

Channels may be intentional (i.e. known about, designed-in, or taken for granted), or unintentional (e.g. electromagnetic interference). In a SoS context, channels form a shifting web of relationships. Interactions over these channels allow component systems to, potentially, cause each other to enter hazardous states.

As with conventional systems, a hazard in a SoS does not necessarily lead to an accident. Chance, system design features, or intentional action may prevent the hazard progressing, or mitigate its effects. For SoS hazards, such prevention or mitigation may occur at the SoS level, by two or more component systems collaborating to remove the hazard. Alternately, it may occur entirely at the component system level, with an individual component system taking unilateral action that either removes its contributing role in the SoS hazard or, if it is itself exhibiting a component system hazard, moves it out of the hazardous state. A large part of SoS safety engineering is, therefore, taking steps to identify hazards, understand their development, and take steps to prevent them occurring in practice.

As an example of a SoS hazard, consider the situation where a UAV on a surveillance mission observes a friendly tank and misidentifies it as an enemy. This is a component system-level failure, and it is the initial *stimulus* (or cause) that changes the state of that component system. The UAV propagates this knowledge over various network links, causing other component systems to believe that there is an enemy tank at the location of the friendly tank. This is a change in multiple component system states, and therefore of the *SoS state*. The SoS is now exhibiting a *SoS hazard* — dangerous misinformation has been distributed through the system.

Eventually, the information about the 'enemy tank' propagates to a ground attack aircraft that is somewhere nearby. The pilot of this aircraft may form an intention to attack the tank. At this point, we have a *component system hazard* — the pilot need only proceed with their normal course of activity in order to damage or destroy the friendly tank. At the point that they do, of course, an *SoS accident* has occurred.

Of course, the example above is simplified; a real system would almost certainly incorporate additional safety measures, such as requiring further target identification (by the attacking pilot

or a third party) prior to an attack. Such 'naked man' SoS examples are of concern, however, because it is the activity of SoS safety engineering that derives such safety measures.

3.4 Automating Hazard Analysis

The literature survey in Chapter 2 covered a variety of attempts to automate hazard and safety analysis. Given the difficulties of SoS hazard analysis described in Section 1.2, the adoption of such techniques is attractive. In order to evaluate the options for this, however, one must first identify what hazard analysis actually requires.

Manual analysis techniques, by necessity, break down the task into a number of steps. This can be illustrated by considering two widely used techniques: Functional Hazard Analysis (FHA) (see Section 2.4.2.2) and HAZOP (see Section 2.4.2.4). A set of steps for each of these, as presented by Pumfrey in [2], were shown in Chapter 2 in Tables 2.1 and 2.3 respectively.

Although the two techniques have different steps, it can be seen that they break down into similar tasks as shown in Table 3.2.

Task	**FFA Equivalent**	**HAZOP Equivalent**
A) Build / acquire model	1	2
B) Enumerate possible deviations of model	2	3
C) Derive effects of deviations	3	5, 6, 7
D) Consider design changes to prevent or mitigate	4	8, 9, 10, 12

Table 3.2: Combined Tasks from FHA and HAZOP

HAZOP steps 1 and 11 appear not to fit into this classification, but this is because they are merely instructions to repeat certain other steps. Unlike FMEA and FHA, HAZOP step 4 is concerned with the identification of possible causes for a deviation. This step is excluded as causal analysis of deviations is not an essential feature of hazard analysis — an analyst does not necessarily have to explain how a particular postulated deviation was derived.

As well as performing those activities that are explicit in the given process being followed, engineers invariably perform additional actions in parallel. They validate the model, questioning the predictions that seem to arise from it. If the model is at odds with their understanding of the world and engineering judgment, they may propose a change in the model, or a change of scale or abstraction (in order to avoid repeatedly analysing equivalent phenomena, or to drill down into detail when some prediction cannot be made at the current level of the model).

These additional activities relate to the overall ability of humans to bring their background knowledge to bear on the hazard analysis problem. An engineer faced with a paper model of a system has a great wealth of knowledge about computers, machinery and human behaviour that allows them to "paper over the cracks" in the model itself. In a manual process, the model is explicitly acknowledged as a proxy for a much more complex reality, which is fleshed out

as needed by the background knowledge of the engineers. By contrast, in a fully automated system the model must be the reality because the system does not have such knowledge.

Given the above descriptions of what activities human engineers perform in hazard analysis, we must now determine whether, and to what extent, these activities can be automated.

An automated system cannot realistically build or acquire a model (Task A). This task is therefore deferred to human intelligence. It is possible, however, to support the development of suitable models through tools and libraries of components. Engineers already use tools to produce a variety of system models, and potentially some of those tools can be reused.

Given a suitable model, can an automated system enumerate the possible deviations of that model (Task B)? Potentially, this is possible, provided that the system is built so as to be able to (i) identify points in the model where deviations can be applied (ii) manipulate the model so as to implement those faults. Given a set of rules about where deviations can be applied, a computer can quickly consider a vast number possible permutations (in some case, all possible permutations), whereas a human would take much longer to enumerate them.

For Task C, the deviations applied in Task B must be applied to the model, and their effects determined. Whether this is possible in any given case depends on the adequacy of the model and the simulation algorithms that can be applied to it. It can be noted that in manual analysis, engineers typically only consider the effect of a single deviation at a time, and for a given deviation they can only reliably predict the immediate downstream consequences when independent of other factors (e.g. environment or concurrent processes). By contrast, although an automated system will not always be able to consider all possible combinations of deviations (because computation time is always a limited resource), or develop their effects for the lifetime of the system, it can project many combinations of deviations over an extended period of simulated time.

In Task D, changes to the design that could be applied to prevent or mitigate the identified hazardous effects must be proposed and evaluated. It is not inconceivable to create an automated system that would attempt to do this, based on a library of historical design changes. Indeed, the HAZOPExpert system [48] has some ability to do this. In practice, however, once an engineer's attention has been drawn to a problem they are generally well-placed to propose solutions. The primary concern addressed in this thesis is that it is difficult to identify all the hazards in the first place.

It was noted above that as well as performing the explicit process steps that are mapped to tasks in Table 3.2, human engineers engaging in hazard analysis will perform a variety of model evaluation and modification actions. All of these are far outside the capabilities of realistic automated systems. There is, therefore, a clear need for human involvement here. As noted above, these actions occur as needed throughout the process. It follows that not only must it be possible for engineers to take such actions, but that they must have visibility of the process sufficient to see when they should take such actions. An analogous issue in general automation is discussed by Norman in [76] — many 'automation failures' occur because of automated

systems that are black boxes, in that they don't allow operators to observe their state or their rules of operation.

It was noted that, in a manual analysis, humans may at any time propose changes to the model being used, either to correct an explicit error or to make a general change, for example to the level of abstraction. Such changes are well being beyond the abilities of any implementable computer system, and human modification of executable and analysable models requires far more work than in the equivalent manual process. Indeed, many transforms of this nature in a manual process are performed solely in the minds of the engineers involved. Consequently, even if a need for a change to an electronic model of a system is identified, it may not be practical to make the change.

3.5 Process Requirements

Drawing on the discussion in the preceding sections, we can summarise the requirements for an effective SoS hazard analysis method. These requirements will be used (in Chapter 7) to evaluate the work.

3.5.1 Requirement P1 — Process must be applicable early in lifecycle

Requirement

It must be possible to perform the analysis (at least initially) during the SoS architecture phase of the lifecycle (see Section 3.2). As well as being presented at a high level of abstraction, such early models are likely to be incomplete and somewhat inaccurate, and the approach must be compatible with these problems.

Rationale

Hazard analysis is most valuable early on in the lifecycle, as the results of the analysis will have implications for later phases of development.

3.5.2 Requirement P2 — Process must integrate with other safety activities

Requirement

There should be a clear interface between hazard analysis and the later parts of the safety process, in particular PSSA. The explicit artefacts that result from the hazard analysis process should be identified, and their uses later in the process traced.

Rationale

The work presented in this thesis is concerned only with hazard analysis. It does not address the confirmatory safety analysis which is necessary to gain confidence of safety and hence create an adequate safety case.

Even within the activity of hazard analysis, the complexity of SoS are such that it is unlikely that any single defined method will provide adequate hazard analysis of all SoS (see Section 5.8.3 for further discussion of this issue). There will always be scope for, and a need for, supplementary hazard analysis techniques that are locally appropriate.

3.5.3 Requirement P3 — Process must support iterative modelling and analysis

Requirement

The process must support embedding in a larger iterative cycle of development, where hazard analysis of the SoS will be repeated many times. This means that it should be practical to change the model to accommodate changes to the SoS design, or in response to criticisms of the model or analysis method, and to repeat the analysis afterwards.

On each iteration it should be possible to modify relationships between model elements, rules describing component system behaviour, and even the level of abstraction at which the model is built. The choice of modelling paradigm and formalism should not rule out too many reasonable concerns.

Once a change has been made to a model, it should be possible to repeat any automated analysis and see results that relate to only the changes in the model, not to random factors or similar uncontrolled influences. For example, many simulation models are stochastic, and the results vary from run to run depending on the pseudorandom numbers that are generated by the computer system. Although it is possible to control the sequence of numbers that are generated by setting the same 'seed' value for each run, a small change to the model may change the order in which those numbers are consumed, which may have a unpredictable effect on the model's behaviour.

Rationale

It is observed by Pyster and Gardner [1] that SoS development is inevitably difficult and iterative. Routinely, changes to one system will impact others. The analysis process must remain responsive to 'change events' (for example, a change in specification of one of the systems) occurring at any time during development. Similarly Sargent, in [77], notes that simulation development, in practice, is likewise inevitably iterative, as the model eventually converges on an adequate approximation of reality.

Regarding criticism of the model, it was noted in Section 3.4 that in conventional manual

methods of hazard analysis making changes to models was easy, especially if the model change did not need to be communicated outside of, for example, a HAZOP team meeting. The nature of the models used in automated analysis makes it impractical to achieve the same level of ease, but overly rigid models will impede the process and (in practice) lead to necessary changes not being made. This, in turn, will lead to misleading results.

3.5.4 Requirement P4 — Process must be feasible and practical to perform

Requirement

The analysis must fit within the budgets and timescales for SoS development, and not take an effort disproportionate to the effort invested in the assembly of the overall SoS.

(Typical estimates allocate around 10% of development effort to safety.)

Rationale

Budgets are invariably finite, and this will inevitably place practical restrictions on the hazard analysis that can be performed for SoS (this may, in turn, go on to restrict the forms of SoS that can be ethically and legally configured and operated, at least in peacetime). It can be observed, however, that hazard and safety analysis budgets can be large, even for relatively small systems. For example Kletz [78] notes that *"the HAZOP on a large project may take several months, even with 2 or 3 teams working in parallel on different sections of the plant"*.

3.5.5 Requirement P5 — Process must provide engineer visibility at all stages

Requirement

At any stage, an engineer must be able to "drill down" and find out *why* the model is producing a particular result. For example, if analysis of the model indicates that a particular set of deviations is causing an accident, it must be possible for an engineer to discover and understand the mechanisms in the model and the analysis algorithms by which the deviations cause that accident. An automated analysis must not be a black box that produces only the final results.

Rationale

As noted in Section 3.4, engineers performing manual hazard analysis techniques are inevitably aware of how their analysis is deriving its results, because they are having to perform all the steps themselves. This provides a valuable insight into both the working of the system and the effectiveness of the method. It is important that this isn't casually discarded in automated methods merely because it presents difficulties.

3.6 The Core of the Approach — Multi-Agent Simulation

In the previous section, it was observed that an effective SoS hazard analysis approach needs to be applicable early in the lifecycle, needs to be practical to repeat in several iterations, and needs to provide visibility of how any particular analysis conclusions were derived. It was noted in Chapter 1 that SoS are difficult to analyse because much of their overall behaviour emerges from the interactions of their component entities. Section 3.1.1 discussed how SoS entities were often characterised by their autonomous behaviour using their own goals, capabilities and knowledge.

In the light of these requirements, the approach developed in this thesis uses multi-agent simulation to perform hazard analysis of systems of systems. Ferber, in [79] provides the following definition of multi-agent simulation: *"Multi-agent simulation is based on the idea that it is possible to represent in computerised form the behaviour of entities which are active in the world, and that it is possible to represent a phenomenon as the fruit of the interactions of an assembly of agents with their own operational autonomy."*

Similarly, Ilachinski, in [68] offers *"[Multi-agent simulations] consist of a discrete heterogenous set of spatially distributed individual agents, each of which has its own characteristic properties and rules of behaviour."*

Typically, the value of multi-agent simulation is asserted in comparison to the mathematical models that have traditionally been used in biology, economics and military analysis. Ferber notes that agent-based models allow the integration of quantitative variables, differential equations and symbolic rules into agent behaviour, thereby providing a means to exploit qualitative observations as well as quantitative information [79]. He also notes that such 'micro-worlds' allow analysts to *experiment* by modifying agent behaviour and adding new agent types, which is not possible with high-level mathematical models. Most significantly for our purposes, Ferber comments that such simulations *"make it possible to model complex situations whose overall structures emerge from interactions between individuals"*.

Ilachinski, in [68] makes a similar point: in a multi-agent simulation, different levels of behaviour can be observed. Analysts can examine both the top-level emergent behaviour and the low-level interactions between individual agents. That is, the simulations can both predict overall behaviour and explain *why* it occurs.

As discussed in Chapter 2, multi-agent simulation has been applied to a variety of military SoS contexts (see, for example, Sections 2.6.3 and 2.6.5), some of them specifically for safety (see Section 2.5.4). As explained in section 2.7, however, the existing work exhibits a variety of problems that make it unsuitable for (direct) use for the hazard analysis with which this thesis is concerned.

3.6.1 Applying Multi-Agent Simulation to SoS Hazard Analysis

When modelling SoS using multi-agent simulation, each component system within the SoS can be represented by an agent. This agent captures the behavioural properties of the component system, and in the running simulation it stores the component system's state, goals and knowledge. The agent communication patterns and protocols can be defined based on the interaction patterns that are expected or desired for the real entities.

Multi-agent modelling provides the potential for models at a range of levels, from the very abstract to the very detailed. Models of differing fidelity can even be combined within the same environment. This supports requirement P1 from Section 3.5 — early-stage analysis can use simple models based on the detail that is available. The ability to combine information expressed in a variety of forms is particularly valuable given the multidomain nature of the knowledge required to built adequate SoS models (as discussed in Section 1.2.5).

The use of multi-agent simulation can show how the behaviour and interactions of known individual agents can cause accidents, thereby supporting requirement P2. Section 3.7.2 describes how the work presented in this thesis integrates with the wider safety lifecycle.

The agents are engineered such that they work together to produce simulated SoS behaviour in a series of scenarios that are believed to be representative of how the real SoS would behave. This simulation model can then be subjected to plausible deviations, and the resulting undesirable behaviour (accidents) observed. Similarly, changes can be made to the standard behaviour of individual agents (e.g. by defining new rules of engagement), or to the overall configuration of the SoS (e.g. by adding extra examples of one component system type). The overall behaviour of the SoS *emerges* from the structure of the entities, their environment, and the mission that is assigned. This helps to address requirement P3.

As noted in Section 3.6, multi-agent simulation (in various forms) is widely used in military and civilian modelling for a variety of purposes. The flexible nature of multi-agent models means that they can often be extended to include new factors, and to produce new types of result. It may therefore be possible to perform hazard analysis using models that also allow analysis of functional requirements or other dependability attributes.

Given the cost of developing adequate simulation models of complex systems, it can be difficult to get support for them from many project stakeholders. If multiple analyses can be performed using the same model, it is likely that more stakeholders will be interested. In addition, there is a growing interest in 'simulation-based' development and acquisition (see Zittel in [80] for an example), particularly for SoS [1]. In simulation-based acquisition (SBA), simulation technology forms the core of the acquisition process and allows acquisition decisions to be informed by the results of simulation-based analyses. In an organisation where SBA is practised, building and maintaining simulation models is a key part of normal operation. This helps to address requirement P4.

Given the observation of a particular high level event (for example, an accident) when the

simulation is run, it is potentially possible to explain it in terms of the behaviour and interactions of individual model agents (although it can be observed that this is a non-trivial activity for complex models). This means that engineers can understand *why* the model produces the high-level behaviour, and therefore assess whether the behaviour is realistic or an artefact caused by an inaccuracy in the model. This addresses requirement P5.

3.7 Process Overview

The process described in this thesis involves an iterative process whereby a simulation is built (using the method described in Chapter 4), the hazards exhibited in that simulation are analysed (using the techniques described in Chapter 5), those results are evaluated, and changes are then made to either the proposed SoS or to the simulation model. The cycle is then repeated, up until the point where the SoS project team are satisfied that they have extracted all the useful information from the model (at least for that incarnation of the SoS). An overview of this process is shown in Figure 3.3.

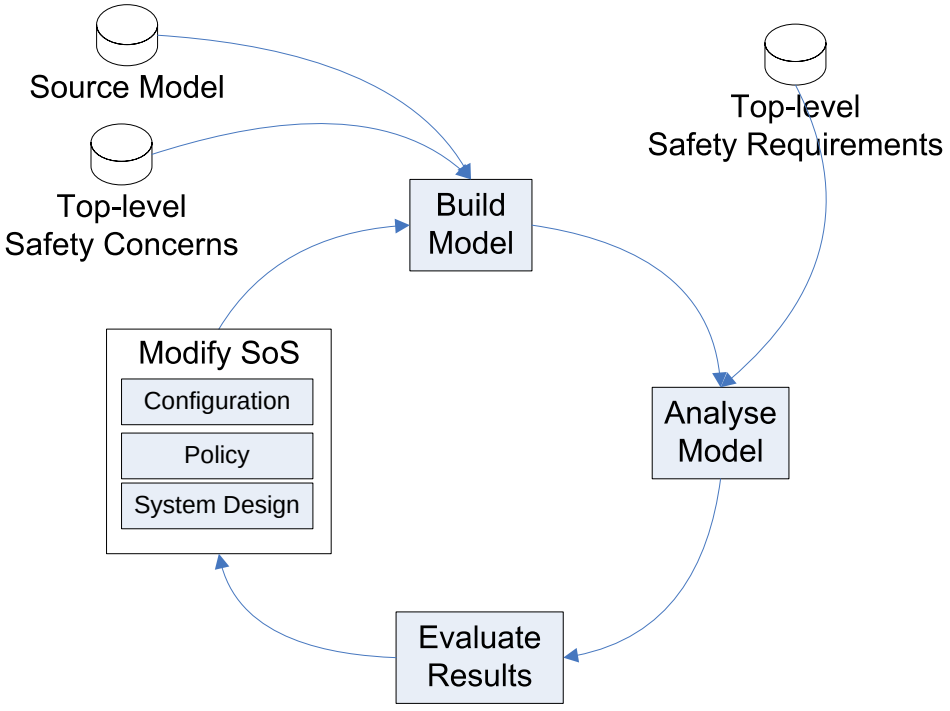

Figure 3.3: Overview of the SoS Hazard Analysis Process

The team may, of course, return to the simulation model at any time to experiment with new system configurations, operational policies or scenarios.

In the double-V SoS lifecycle, as discussed in Section 2.2.1, the systems engineering activities continue throughout development. The simulation model will be progressively improved over time and, additionally, the concerns that the model needs to capture (see Section 4.5) are also

likely to change over time, thereby (partially) invalidating any model that was based on an earlier set.

For clarity of exposition, the presentation of the process in this thesis is essentially linear, and is most directly applicable to the first iteration where the simulation model is being built and analysed for the first time. Later iterations can be expected to become progressively less thorough, with the emphasis being on review and revision of the work performed in the earlier iterations.

3.7.1 Role of Human Engineers

Although part of the process described here is automated, human safety engineers still have a number of important roles to play in it. It is obvious that engineers are responsible for building, maintaining and updating the model. Less obviously, humans must bring engineering judgement to the process above and beyond the explicit steps. They may critique the model at any points, discovering possible flaws and making suggestions for possible future improvements. For example, Section 4.6 discusses the role of domain experts in providing supplementary information to resolve identified gaps in a model.

When analysing the developed model engineers must validate the results of the analysis using a variety of methods, including comparison with other sources of information. This is discussed in Section 5.8. They must also translate the results into a variety of forms to enable communication of the results to a variety of system stakeholders, as described in Section 5.11.

This topic will be addressed throughout the description of the process in Chapters 4 and 5, and a variety of tasks that require engineer skill and judgement will be identified and described.

3.7.2 Role of the Process in the SoS Safety Lifecycle

The value of a simulation-based hazard analysis process can be better understood by putting in the context of the SoS safety lifecycle as presented in Section 3.2.

In the initial *accident identification* phase, the simulation approach has little to contribute, although the results of this phase are critical because it is those accidents that will be detected in simulation later. Similarly, the output of the *requirements analysis and specification* phase will not provide sufficient detail for effective simulation-based analysis.

Once development progresses to *architectural design*, however, simulation analysis can be of value in *hazard identification*. It is common for a proposed SoS to have a Concept of Operations (CONOPS) available early in the development process, providing a conceptual model that describes the entities in the system and the ways that they are expected to interact. Architectural models are presented at this stage to support and expand those views. In this thesis, MODAF architectural models are used; see Section 2.6.2 and Chapter 4. These early models can be used to perform hazard analysis, and the results of such analysis is valuable because knowledge at this early stage in the lifecycle can efficiently guide later development.

This thesis is primarily concerned with the architectural design phase, and the case studies presented in Chapters 4 through 6 start from architectural models.

As the lifecycle progresses to *detailed design*, additional information about SoS structure and behaviour can be incorporated into the model. Section 4.6 shows how such information can be used during the modelling process. This increased detail (such as failure probabilities) supports the use of the of the simulation in quantitative *risk assessment*.

As specific systems are designed or identified for the roles in the architecture, the simple models used initially can be replaced by more sophisticated and accurate ones. It can be observed that high-fidelity synthetic environments (SEs) may be useful at this stage, and provide facilities to reproduce SoS and their component systems at a high level of detail. However it is possible, even with simple models of component system behaviour, to elicit patterns of overall SoS behaviour that are potentially hazardous. The analysis method presented in this thesis can be used to find possible hazards that can be explored further in an SE, and any safety-relevant information gathered from more detailed SE representation can be integrated back into the hazard analysis model.

As SoS *integration* proceeds, it becomes possible to partially validate the dynamics observed in the model against the implemented and integrated SoS. The wider effects of problems observed in the integrated SoS can be explored in the simulation, with the potential as always to identify new hazards. For example, communications problems that are observed between two component system types can be replicated in the simulation model, possibly revealing new hazards stemming from those problems. Similarly, problems identified during *testing, validation and verification* can be evaluated for their safety impact.

The existence of the simulation model and analysis approach allows models of a component system under development to be tried in the overall SoS context (as captured in the simulation model) in order to evaluate the impact of design choices on safety behaviour. In effect, the context of the system is provided by the SoS owner, allowing the component system owner to use the simulation model to discover (a) the influences that the context will have on their system and (b) the effects that their system will have on the overall SoS. Should the latter result in a hazard at the SoS level, the SoS owner may specify further safety requirements on the component system to ensure that it does not produce those effects.

Should a component system owner discover a new hazard in the SoS through their experimentation with the simulation model, they can provide this information to the SoS owner who can study it further. The SoS owner remains, however, responsible for the later analysis of the whole SoS with all component systems in context.

During the integration and testing phases, identified hazards are removed or mitigated by a variety of means, such as equipment modification or the imposition of operational policy (for a discussion of the the latter, see Hall-May in [81]). Then, when producing the *safety case*, an argument is made that these changes were successful, and that the identified hazards are no longer of concern. Furthermore, the use of the technique itself can be used to support the claim

that all hazards in the SoS have been identified — see Section 5.13.

More speculatively, a high-speed part-automated analysis could be useful during the *deployment* and operational phase, by providing a way to explore the safety impact of component system loss or change during SoS operation. For example, the analysis could derive the implications of the loss of a UAV that recently occurred, given scenarios that are likely developments of the SoS's current situation.

It is possible (perhaps likely) that as the SoS design and the associated model matures through the lifecycle, the set of hazards revealed by analysing the model will increase. This represents an improvement in the analysis, not necessarily a problem with the system.

To summarise, the use of multi-agent simulation models for hazard analysis allows the representation of safety-related knowledge about the behaviour of the SoS, allows the consequences of that knowledge (in terms of the safety of the SoS) to be derived, and allows for the progressive integration of further and better knowledge as the SoS lifecycle progresses.

The use of the results of the analysis in the lifecycle is further discussed in Section 5.12.

3.8 Running Case Study — Anti-Guerrilla Operations

In order to illustrate the details of the process presented in Chapters 4 and 5, an example case study will be used. This case study deals with a hypothetical military system operating in an anti-guerrilla role. It uses network-centric technology to coordinate (unmanned) mobile artillery and airborne special forces in action against an enemy who are hard to detect and spread across a wide area.

An overview of the system's operation in shown in Figure 3.4. There are four key parts to the unit: unmanned air vehicles, unmanned self-propelled guns, transport helicopters and special forces sections. The standard operating procedure is that UAVs locate targets, the guns provide artillery bombardment to suppress, disorder and weaken the target, then the special forces move in (carried by the helicopters) to deal with any remaining enemy and secure the area.

Only a single vignette is described for this scenario. The UAVs patrol the area, using a variety of sensors to observe any activity in the terrain below them. When they detect signs of enemy movement, they communicate this to the other component systems via some form of data fusion, and this allows the guns to open fire. The UAVs continue to monitor the situation, and feed their observations back to the other component systems. When it appears from the shared picture that the enemy have been adequately weakened the guns will cease fire and the helicopters will move in.

The AGO SoS is a good example of a simple SoS. It has an *overall goal* (eliminate the enemy positions). This is achieved collaboratively by fulfilling the *individual goals* of *distributed*, *mobile* component systems. For example, the enemy positions are initially located by the reconnaisance objectives of the UAV. The component systems react *autonomously* to their obser-

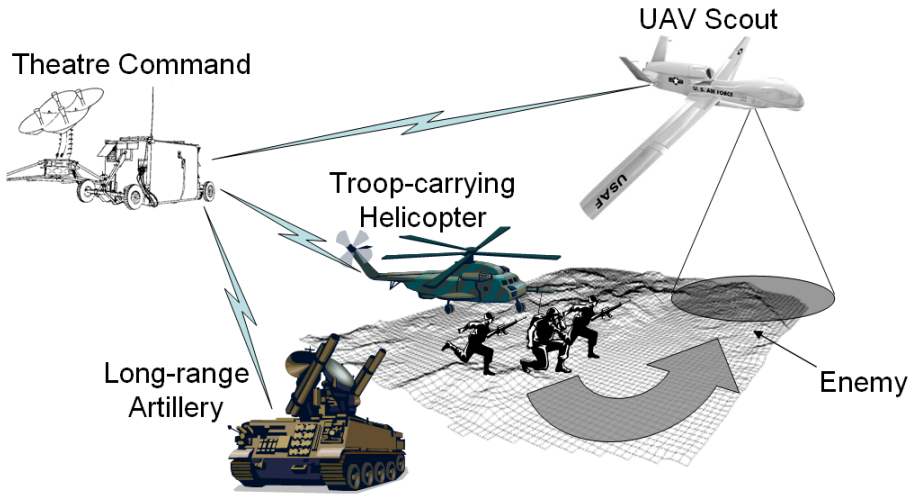

Figure 3.4: The Anti Guerilla Scenario, reproduced from [4]

vation of the world (both directly and via multi-platform data fusion) rather than being driven by centrally-coordinated orders. No one component system type has the *capability* to complete the mission on its own, so the different component systems have to *collaborate*.

As the AGO scenario is used only as an example, its design artefacts are presented only partially, as needed for illustration of the method. A full case study is given in Chapter 6, with its corresponding artefacts in appendices C and D.

Chapter 4

Building Simulation Models for SoS Hazard Analysis

This chapter presents requirements and a method for producing simulation models of systems of systems for the purpose of hazard analysis. The resulting simulation models will be capable of exhibiting the accidents that are of concern, and will define a set of deviations that allow analysts to explore the behaviour of the system under a variety of abnormal conditions.

4.1 Overview of Modelling

4.1.1 Choosing What to Model

As was observed in Section 2.6, a model is always an abstraction of the (perceived) relevant aspects of some real or conceptual system. Much of the detail and complexity of a real system, or much corresponding detail that could be proposed for a conceptual system, will be excluded from the model. The core question in modelling, then, is "What do we *need* to model?" or, alternatively, "What can we ignore and still reproduce the aspects of the system's behaviour that we need?".

For the case of hazard analysis, the answer to the first question is "Whatever can express our failure hypotheses". In order to know this, an engineer developing a SoS model will therefore need to know what failure hypotheses they want to explore. These will vary from case to case, system to system, but they will all be of the form "(a particular set of deviations from the SoS's normal configuration) causes (a particular accident)". For example, in a civil aviation SoS, one might want might be concerns that the loss of primary radar might cause midair collisions during landing or takeoff operations.

4.1.2 Representing the Real World

The process of modelling involves translating from an expression of the system and its context in terms of the real world and in the terminology of domain experts into an executable simulation model. Drogoul et al., in [5] note that this involves a set of distinct *roles* taken by the participants in this process, with each role having a distinct skill set that they bring to the process. The roles identified by Drogoul are the *thematician* (more commonly known as the *domain expert* or *subject matter expert*), the *modeller*, and the *computer scientist*. It is possible for a single individual to perform all three roles, but in practice it is unlikely that any given individual will have optimal skill in all three areas.

Drogoul views the multi-agent modelling process as a process of moving from *micro-knowledge* (descriptions of the agents in the system and their individual behaviours) to *macro-knowledge* (a description of the overall system behaviour that emerges from the actions and interactions of those agents). Her description of the multi-agent modelling process is shown diagrammatically in Figure 4.1. She describes three development stages prior to final implementation.

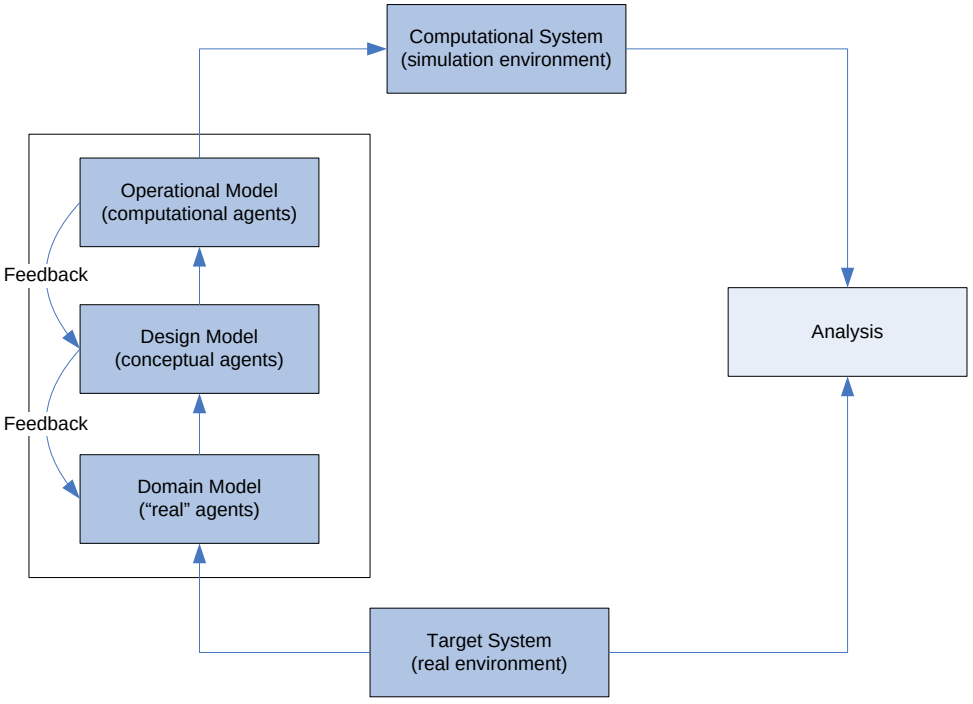

Figure 4.1: The Multi-Agent Modelling and Simulation Process (from [5])

The *domain model* is a model of the system using assumptions and terminology based on knowledge of the real-world SoS and the environment in which it operates (or, for purely conceptual SoS, based on knowledge of other, similar, systems). It is produced by a *domain expert* who has good understanding of the system domain (for example, military operations) but perhaps little understanding of multi-agent modelling or computer simulation. This model is needed because it captures the information about the real world that the later models will be

based on.

The *design* model expresses the system to be modelled in multi-agent systems terms, using concepts from that domain. Some examples of such concepts are shown in Figure 4.2. Producing the design model is the role of the *modeller*.

A model of a SoS expressed in terms of multi-agent concepts

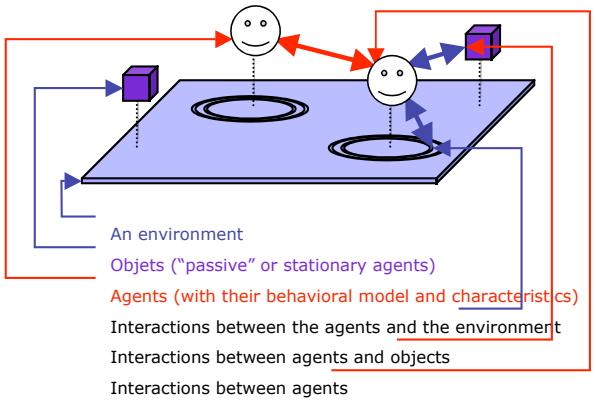

Figure 4.2: The Information Found in the Design Model (from [5])

The *operational model* contains a description of agents in terms that can be directly translated to an imperative computer program; Drogoul calls this *"agents as an implementation technique"* [5] and assigns its development to the *computer scientist* role. Drogoul motivates this model on the basis that precise technical properties (such as the mechanism used to schedule agent actions) can have a significant effect on the results that are derived from the model.

Each model is developed in order, as shown in the diagram, based on the model developed in the previous stage. For example, the domain model is used to derive the design model. Implementation (as a computer program) follows directly from the operational model, which by this stage should be in sufficiently precise terms to make this possible.

The significance of these stages, and the importance of performing each stage well, stems from the fact that the quality and correctness of the transforms between the stages determines how the results that are derived from the final, implemented simulation model relate to the domain expert's concept of how the real-world system works. Drogoul states that if discrepancies between the views of the various roles are not made explicit and resolved to the agreement of all parties, *"there is absolutely no guarantee that what is being designed and implemented corresponds to what has been desired and modelled by the thematician"*.

The general question here is "How can one derive an implemented simulation model from a domain model?". This thesis will not attempt to resolve the general question; it will address only the specific issue of developing multi-agent simulation models for the purpose of hazard analysis of SoS.

4.2 Requirements on SoS Simulation Models

Given the nature and characteristics of SoS and SoS hazards, as presented in Chapters 1 to 3, the following requirements can be identified for simulation models of SoS.

4.2.1 Requirement M1 — Models must adequately represent SoS

Requirement

The model that results from the modelling process must adequately model the system, in terms of what it contains and how it behaves. As the system contains people (see Section 3.1.3), the model must be 'behavioural' (i.e. it encodes how humans actors are expected to behave in the modelled situation — see Mitchell in [82]). It is important, too, that the description of how the system behaves be sufficient to determine its behaviour under a variety of non-standard conditions, including failure of technological systems or aberrant behaviour by humans. Mitchell, in [82], refers to models that support such alteration of causes and effects as 'exploratory'.

A corollary of this is that the modelling framework needs to express those characteristics of a SoS that make it uniquely a SoS (see Section 3.1.1 for an enumeration of these characteristics). For each characteristic, an argument needs to be made that the modelling approach and notation supports that characteristic adequately.

It is important to discuss not only whether the modelling approach and notation can express the SoS characteristics, but whether it does so adequately for the purpose at hand. It is not in question whether non-specific modelling approaches (such as the UML software notation [83], network modelling (see Kotov in [27]), or SysML [84] *can* be used to represent SoS); rather, it is the contention of this thesis that, for the purpose of hazard analysis, they will do so only poorly.

For each characteristic identified in Section 3.1.1, and given its potential contributions to SoS hazards from Section 3.1.2, some specific requirements for representation in a model can be drawn out. Table 4.1 shows the modelling requirements imposed by each SoS characteristic.

It is also important that it is possible to build models that are capable of recreating actual historical systems of systems accidents — an inability to model the circumstances of such an accident (and to reproduce the historical result as at least a possible outcome) indicates a deficiency in the modelling approach. The characteristics that are implied by several historical accidents are presented in Table 4.2.

Rationale

If the behaviour of the system is not modelled, or is not modelled in a sufficiently general way, then it will not be possible to analyse it. Hazard analysis will hence be impossible. For example, simple 'script' representations, such as a UML sequence diagram, capture the system

Characteristic	Model must be able to express…
System goals	Overall system objectives
Multiple components	Individual entities
	Loss of components during operation
	Component use of shared resources
	Use of components in tasks
	Distinction between components and roles
Component SA/worldview	Individual component SA. Individual coordinate systems
	Fallible world sensing
Component goals	Local goals
Component capability	Different capabilities (kind and degree) of different entity types
	Explicit sensors and actuators that can fail
Geographical distribution	Location of entities in physical space
	Component sensing, local SA
	Terrain and its effect on communications and sensing
Mobile systems	Entity movement
	Terrain effects on movement
Autonomy	Independent action loops
Collaboration	Shared world (i.e. the effect on the world by one component can effect, and be sensed by, others)
	Deviation of plans/behaviour sequences
	Exchange of imperative statements
Communications	Message and data exchange between entities
	Fallible communications hardware
	Jamming, weather
Ad-hoc communications	Dynamic change of radio nets (leave or join at will)
	Discovery of networks
Heterogeneity	Compatibility (or not) of communications hardware

Table 4.1: Implications of SoS Characteristics for Modelling Requirements

behaviour in a single expected case, but they don't make it possible to change one part of the system and observe the consequences. An analyst can propose deviations to entity behaviour early in the script, and project the effects of those deviations on later events in the script. The notation does not help them, however, explore the potential *changes to the sequence of events* caused those deviations. Hazard analysis cannot be performed, directly, using such a model. A more general description is therefore required.

If the model doesn't adequately represent SoS characteristics, then the advantage of an 'SoS-specific' modelling approach is unclear; a generic modelling technique might have been more appropriate.

Finally, the investigation of past accidents is our primary source of data for determining how accidents happen. They represent the primary data source against which our understanding of system safety must be evaluated. Hence the modelling approach must be able to reproduce such accidents, so that such an evaluation is possible.

Although the (fortunate) infrequency of serious SoS accidents prevents us from performing true knowledge-based safety analysis (e.g. as described by Mahmood in [85]), the historical record is a vital source of guidance. If a modelling approach cannot recreate a given historical

Accident	Source	Important Factors
Black Hawk friendly fire	See section 1.1.1	Equipment failure (IFF mode IV)
		Configuration errors (IFF codes)
		Effect of terrain (loss of radar contact)
		Cognitive effect of expectation (F15 not expecting to see allies)
		Limited perception (incorrect visual identification)
		Limited memory (AWACS 'lost' the Black Hawks)
Lake Constance TCAS	Accident report [61]	Heterogeneous (effective) policies
		Limited perception
Various friendly-fire accidents	See Regan in [14] for numerous examples	Erroneous expectation of peer position
Afghanistan GPS	See section 1.1.3	Unexpected equipment interface behaviour
		Timing between actions and events
USS Vincenz	See section 1.1.2	Implementation details of IFF system
		Effects of cognitive load and combat pressure

Table 4.2: Requirements Implied by Historical SoS Accidents

accident then it is excluding at least a small class of accidents from analysis.

4.2.2 Requirement M2 — Models must capture realistic SoS contexts

Requirement

The scenarios that are modelled and explored should reflect actual missions that the SoS will be tasked with and situations that the SoS will be placed in. It is likely that many such scenarios will come from source models, but it is important to go beyond that and confirm that a representative set of scenarios has been elicited.

It follows that the model must adequately capture the operating environment of the SoS. The environment model should include all relevant factors, such as terrain, weather, and opposing forces. It can be noted that, given the complexity of possible environments and the wide range of environmental properties that could be modelled, determining what this set of relevant factors is is non-trivial.

Rationale

A complex SoS may be highly sensitive to its context, and it follows that a thorough investigation of the SoS's behaviour will require a comprehensive exploration of the contexts that it will encounter. A set of scenarios that has been produced for other, non-safety, analyses may not be adequate because effective hazard analysis requires a representative set of situations, and existing scenario sets are unlikely to have been developed for this explicit purpose, particularly in the early stages of development. More likely, scenarios provided at the early stages of SoS development will explore SoS performance in a few 'typical' situations, or demonstrate particularly interesting or novel capabilities of the SoS.

It can also be noted that analysing *unrealistic* or wildly improbable situations will present engineers with irrelevant distractions, and so waste time and other resources. Hence the need to evaluate the relevance of scenarios considered.

The environment model captures the stimuli that the agents in the SoS are exposed to, and is therefore required in order to derive how the agents will actually behave in some situation — the behaviour of an agent is derived from both its own properties and the properties of the environment it finds itself in. Furthermore, all external agent actions, and any consequent interactions between agents, must take place via the environment. Such actions and interactions will be constrained and modified by issues of agent proximity, perception and communication, all of which are aspects of the environment to be modelled.

It can be noted that much of the treatment of agent environments in the literature is simplistic. Ferber, for example, in his otherwise detailed textbook [79], restricts the discussion to centralised environments with no concept of spatial position.

4.2.3 Requirement M3 — Models must incorporate plausible deviations

Requirement

The modelling process must lead the modeller to consider deviations of possible expected SoS behaviour. It is therefore necessary to deviate both the SoS itself and the scenarios that it is expected to encounter. This should be conducted using a systematic, reproducible approach that guides the modeller to consider those deviations that are likely to cause accidents and to implement them (as options) in the model. It must be possible to make some kind of claim about how the set of deviations that the modeller has derived is acceptably complete, i.e. that it is likely that any deviation likely to be of consequence in the real SoS has been identified.

Rationale

As explained in the literature survey (see Section 2.4), the process of hazard analysis is inevitably based on identifying the deviations of expected component system behaviour that could lead to hazards. Simple 'brainstorming' techniques can be effective here, but explicit,

systematic techniques can prompt modellers to consider deviations that they otherwise might have missed. Such systematic techniques can capture accumulated expertise and knowledge from past experience with similar systems.

The ability to claim completeness is important because hazard analysis provides the starting point for safety analysis (see Section 2.4.1), and performing adequate safety analysis requires that all credible hazards have been identified. Building a risk-based safety case also requires that an argument be made for the completeness of the set of hazards identified. If a hazard analysis process cannot provide the grounds for this argument, then the safety engineers may not be able to do demonstrate that the SoS is safe to operate.

4.2.4 Requirement M4 — Models must provide the features needed for analysis

Requirement

The detail that is included in the model must be appropriately tuned to the requirements of the specific deviation hypotheses that we wish to explore during the analysis. A deviation hypothesis relates a particular deviation of the model to a particular accident.

For example, if the modeller is concerned that two vehicles could collide because of poor visibility in bad weather, then the model needs to represent appropriate detail of vehicle movement, vehicle (or driver) perception in different weather conditions, and the spatial positions of the vehicles.

It must be possible to show that the aspects of the system that are of concern to the modeller are represented in the model in all the forms that it takes during the modelling and simulation process, including the final executable simulation and the output that it produces.

Rationale

It is impossible to model everything, and likely to be counterproductive to try. A model, simulation or otherwise, is always a simplification (through abstraction) of the real system that it represents (see, amongst many others, Weinberg in [86]). The disadvantage of missing detail is obvious — an analysis, especially and automated one, cannot reveal a hazard that occurs in a mechanism that is not explicit in the model.

Excessive detail, too, is problematic — it increases model development effort, computation time to perform exploration, and most likely also the ‘noise’ in the resulting data that analysis techniques (manual or automated) have to deal with. Hoeber, in [87], remarks on how a small increase in the number of parameters presented by a model can dramatically increase the time needed for an automated exploration of the search space. Also, as noted by Lauren and Stephen in [57], greater detail does not necessarily mean greater fidelity; an extremely complex model can still be wrong, and its very complexity makes it harder to carry out adequate validation.

As Droghoul noted (see Section 4.1.2), the stages from source model to implementation involve

several distinct skill sets (and often individuals) and this provides ample opportunities for errors and distortions to be introduced into the model. There must therefore be an auditable trail that can be followed to ensure that, at least for the most critical aspects of the model, such distortions do not prevent the model being useful for hazard analysis.

The nature of hazard analysis models (and, indeed, safety models in general) is that they are extremely difficult to validate. It *is* possible to 'close the loop' by comparing the results of simulation runs with the scenarios that were provided in the source model. If the sequence of events occurring in the simulation model is adequately similar to the sequence of events expected in the scenario, then some confidence can be gained that the model reproduces the expected behaviour of the real system. This, of course, is unsatisfactory as it will only explore a small part of the system's behaviour space; in most cases, the source model will only describe normal-case scenarios. After all, if the source model already described all deviated-case scenarios, then hazard analysis would not be necessary (although confirmatory safety analysis would). In any case, finding errors at such a late stage is very expensive compared to identifying them early in the SoS lifecycle.

4.2.5 Requirement M5 — Models must be verified and validated

Requirement

Any model, if it is to be used, must be verified (against other descriptions of the system) and validated (against "the real world"). The simulation model developed for hazard analysis should be verified against the original source model and against the conceptual views of the system that are held by domain experts, particularly those involved in system development. It should also be verified against the emerging design and policy documents as the system concept develops over time.

It should be noted that apparent inconsistency between the simulation model and other sources does not necessarily mean the simulation model is wrong; it may be the other model or authority that is in error. Such discrepancies, however, must be justified; if a justification cannot be made, then it is a cause for concern.

There are a wide range of approaches to simulation model validation; a brief review of techniques is given by Sargent in [88]. Although the nature of hazard analysis in complex sociotechnical systems (especially at an early stage in the lifecycle) means that many candidate techniques are inapplicable (such as validation against historical data or by using the model to predict the behaviour of the actual system), it is important that an analyst seeking to introduce a new simulation-based technique in place of established manual analysis methods be able to argue that the models used by their new approach is at least as valid as the models used by the existing manual techniques.

Rationale

A model that is unverified and unvalidated may mislead analysts and engineers, and either waste time and resources on fruitless investigation of unrealistic hazards or cause them to miss an important hazard.

As noted above, inconsistencies between models are often acceptable (and to some extent unavoidable). Unexplained inconsistency, however, will confuse engineers, make the system difficult to validate, and may conceal hazards.

4.2.6 Requirement M6 — Models must be amenable to the desired analyses

Requirement

The model must be amenable to analysis by the particular techniques that are to be used on it. Clearly, a model that provided no deviations, which provided no output and which could not be observed in any way would be unsuitable for any forms of hazard analysis. However, in practice, the specific requirements on a model will depend on the analysis technique in question.

It is not sufficient for the model to make analysis with a given technique *possible*; care must be taken that the resulting analysis is defensible. It is also important that the model supports the repeatability of any given analysis after change.

Rationale

There is little point in having a model that can't be analysed. Manual analysis techniques are often highly flexible in terms of their ability to make some use of an inappropriate model, but as noted in Section 2.4 manual techniques are not practical for SoS hazard analysis.

For the issue of quality of results, automated analysis techniques are entirely dependant on the quality of their input, and therefore problems with the input will lead to problems with the results of the analysis.

4.3 A Modelling Process

Given the requirements above, this thesis proposes a modelling approach that meets them. Consistent with Drogoul's theory of multi-agent modelling (see Section 4.1.2) the process is presented as a series of stages by which the original source model (MODAF, in this case) is transformed into the executable simulation model, with identified individuals with different skill sets being needed at each stage. The resulting executable simulation model produces output suitable for the analysis methods proposed in this thesis.

This is supported by the providing a set of explicit cross-checks at each stage that guide the

modeller to check the results of that stage against earlier stages. First, consistency checks between domain model and the developing design and implementation are specified (see Section 4.13.2), which prevent 'drift' of the model at a global level. This is potentially sufficient to ensure that later-stage models are consistent with the source model, but here it is supplemented with cross-checks between one stage and the preceding stage (see, for example, Section 4.8.3). These latter checks prevent local 'drift'; they remove the risk of each modelling stage introducing subtle (perhaps unintentional) changes to the model. This helps to support requirement M5.

In support of requirement M4, the process presented in this thesis guides the modeller to explicitly identify the concerns that need to be explored in the model. These concerns are identified at the start of the process, working with the source model, and then are traced through each transform, checking at each stage that they are still adequately supported by the emerging model. In effect, the concerns weave an annotated 'thread' through all the stages in the development of the simulation model, allowing the final implementation to be validated in terms of its support for the identified concerns. This ability to validate the emerging model supports the satisfaction of requirement M5.

It is possible, while following this process, to use other forms of analysis at a lower level in order to improve the simulation model. Examples of this include analysing the internal structure of an agent in order to determine how it will behave in response to certain stimuli, and analysing the low-level implementation details of an electronic communications protocol in order to derive entity-level deviations related to it. This helps to address problems with the level of detail of the model, and thereby helps to address requirement M4.

4.3.1 Process Steps

The steps of the method are:

1. Acquire a MODAF source model for the system under analysis
2. Identify key concerns that need to be captured by the final simulation model
3. Identify and acquire necessary information that is missing from the source model
4. Decompose the scenarios provided by the source model into constituent vignettes
5. Transform the domain model that has been developed in the previous steps into a design model
6. Define deviations for the agents in the design model
7. Define deviations for the vignettes
8. Trace the identified concerns to the deviations, and verify that adequate coverage is achieved

9. Transform the design model into an operational model

10. Implement the operational model as an executable simulation

These steps are explained in the sections that follow. For each step, an illustration is given via its application to the AGO case study that was introduced in Section 3.8.

4.4 Step 1 — Acquire MODAF Model

The first step in the modelling process is to acquire a MODAF model (see Section 2.6.2) of the system that is to be analysed, and development of the executable model proceeds based on this. The MODAF model captures the domain expert view of the SoS at the time of development. It captures the component systems of the SoS, their communication and (explicit) cooperation relationships, the functions or capabilities that they provide to the SoS, and their behaviour patterns. This relates to model requirement M1.

The MODAF model is referred to as the 'source model' as it is the primary source from which later models will be derived. It is not, however, the 'domain model'; that term will be used to refer to the combination of the MODAF model with additional information as described in the first few steps of the process.

Standard MODAF models offer a level of granularity that is suitable for the analysis we want to perform — a MODAF 'operational node' corresponds closely directly to a SoS 'component system' and therefore to an agent in an executable multi-agent model. MODAF is an architecture model and the hazard analysis in this thesis is, at least initially, applied at the architectural level.

The use of a MODAF model also allows the modeller to exploit the domain expert's knowledge about appropriate level of detail and granularity — the MODAF model provides a default division of the SoS into operational nodes. It can be seen that this goes some way towards satisfying model requirement M4.

It is likely that, at least for SoS developed by the UK military, that a MODAF model will be available early on in the SoS lifecycle. This is significant because there is a need to produce hazard analysis simulation models early in the SoS lifecycle (this was noted in requirement P1 — see Section 3.5.1). Typically, few models are available in the early stages of any system, but as noted in the MODAF Executive Summary *"the option of a rigorous architectural approach in the early stages of the system lifecycle when there is still an opportunity to correct issues regarding system context, scope, interfaces, etc is likely to have a high impact on this EP* [Equipment Programme] *risk and should realise good savings"* [64]. It is therefore likely that MODAF models will be available. Questions about the quality, accuracy and comprehensiveness of MODAF models (see Section 2.6.2) can be weighed against this advantage of early availability.

The modeller must analyse the quality and completeness of the model they acquire. For example, they must verify that all necessary products are present and that all the operational nodes have been developed (for example, to include their participation in the sequence diagram of OV-6c). The absolutely essential core products are OV-1 (which provides an overview of the SoS and its expected context), OV-2 (which shows the component systems in the SoS and the communications relationships between them) and at least one of OV-5 or OV-6c (which show the sequence of actions, and interactions, that the SoS entities are expected to perform in a given scenario). Beyond these, the presence of additional products will provide further detail that will be valuable for the modelling effort.

4.4.1 AGO Case Study

A partial MODAF description of the AGO system is given by Despotou in [89], and reproduced in Appendix A. Products OV-1, OV-2, OV-5, OV-6c and OV-7 are present, along with SV-4 and SV-5. This is a reasonably complete set of products, although there is no OV-6a or 6b — there is information on sequences of events at the SoS level, but lack of detail on the behavioural rules and states of individual nodes. The case study will therefore illustrate how one can work around a lack of such detail. All the operational nodes that are identified appear to be represented in all the appropriate products. One scenario is provided.

The model presented in some of the artefacts, particularly OV-2 and OV-6c, is inconsistent with the description given earlier (in Section 3.8) — it is overly centralised, and shows a central 'Mission Command' entity driving the action of the system through direct orders. To make the system suitable for use as a case study in this thesis, revised versions of OV-2 and OV-6c were produced that show the component systems cueing most of their actions autonomously from the results of data fusion, with a greatly reduced dependence on the central command entity. These revised versions of the products are presented in Appendix B.

It can be observed that there is no obvious way to show, in MODAF, how continuous decentralised data fusion provides a Combined Operational Picture (COP) for all participating entities, or to indicate that the component systems in the AGO SoS react autonomously to changes in the COP (as opposed to responding directly to explicit messages coming from other entities). It is likely that COP maintenance would occur via a continuous process, rather than by explicit irregular messages, but the nature of the sequence diagram notation forces the latter representation if it is to be comprehensible. In Appendix B, the MODAF products are annotated to indicate those interactions that would occur via continuous data fusion — see the comments that follow each diagram.

4.5 Step 2 — Identify Concerns

The modeller must identify the concerns they have regarding the general categories of unsafe behaviour that could occur in the system, and that consequently need to be represented in all

models including the final implementation.

The first category of concerns is accidents. Deriving these is fairly straightforward; the set of accident types ('collision', 'friendly fire', etc.) was identified in Section 3.3. For all friendly entities in scenario, the combinations of these with accident types can be enumerated.

For example, if we have two helicopters and a ground vehicle, all armed, then we can have accidents involving the two helicopters colliding, a helicopter landing on/colliding with the ground vehicle, friendly fire between the two helicopters, friendly fire by the helicopter on the ground vehicle and friendly fire between the ground vehicle and a helicopter. It is likely that we would add a more generic concern of 'subject to enemy fire' to each entity.

The second aspect of identifying concerns is to identify the kinds of deviations that we believe might cause the system to have an accident in a situation that would be safe in the normal, undeviated case. This is primarily a task that requires analyst judgement and the use of prior experience, although consideration of the source model may provide a stimulus. For example, a common top-level concern is 'unreliable radio communications'. Study of a MODAF model (particularly OV-2, 'Operational Node Relationships Description') may help to identify pairs of component systems where a loss of radio communication would be particularly significant.

4.5.1 Relating Concerns to MODAF Model

Once a set of concerns have been identified, the modeller must relate them to the MODAF model, identifying the concerns that have particular implications for particular MODAF products or operational nodes. Table 4.3 suggests the MODAF products that are relevant for several types of concerns. In general, each concern should be mapped to specific operational nodes in the MODAF model, and hence to specific component systems within the SoS.

Type of Concern	**MODAF Products**
Communications	Information links are shown in OV-2, and specific communications interactions are shown in OV-6c.
Capabilities	SV-4 describes system functions, SV-5 relates these to operational activities, and OV-5 and OV-6c show the timing and circumstances of those activities.
Interaction with the environment	Some reference to the environment is made in OV-1, and some environmental interactions may be shown in OV-5 and OV-6c.
Accidents	Accidents are not generally explicitly represented in MODAF, although actions or capabilities that could make them possible may be present in any product.

Table 4.3: Guidance for Relating Concerns to MODAF

For example, for the concern 'unreliable radio communications' the modeller could note that deviations of radio communication impact those needlines in OV-2 that would be served by radio communication, and those interactions in OV-5 and OV-6c that correspond to radio mes-

sages.

4.5.2 Recording Concerns

Once the modeller has identified the concerns, the concerns should be recorded in table form so that they can be referred to throughout the modelling process. At this early stage, the concerns are expressed at a very high level, using domain terms and references to the MODAF model. A format that can be used for this is given in Table 4.4.

ID	Type	Description	Model Significance	MODAF Significance
1				
2				

Table 4.4: MODAF-level Concerns Format

4.5.3 AGO Case Study

Four example concerns for the AGO example are developed in table 4.5. It can be observed that MODAF does not provide particularly good support for the expression of accidents — as emergent effects of system behaviour, they are below the level at which MODAF is generally used. Low-level details such accuracy of the artillery are not present in the AGO model, although potentially they could be provided in some of the SV products. This lack of coverage means that the 'Model Significance' is important for recording what needs to be represented to capture the accidents.

4.5.4 Scope and Completeness of Concerns

It is not intended that the modeller try to identify concerns corresponding to *all* potentially important aspects of the system to be modelled. Rather, they should restrict themselves to modelling just those aspects that are *necessary* for the system to be analysed effectively.

The modelling process presented in this chapter has been developed to guide consideration of many of the model aspects that are important for SoS hazard analysis. When identifying concerns, the modeller need only identify concerns that are of particular system for the system they are modelling or for the purposes for which it will be used. Judgement of what these are is left, primarily, to the skill of the modeller.

4.6 Step 3 — Identify Missing Information

After identifying concerns as described in the previous section, it may be apparent that the MODAF being used as a source model inadequately describes the system and its environment.

ID	Type	Description	Model Significance	MODAF Significance
1	Accident	Collision between a helicopter and a UAV	Helicopter spatial position at different times, UAV spatial position at different times	Helicopter movement indicated by 'Transport Special Forces' activity in OV-5 and OV-6c. UAV movement indicated by 'Patrol Area' activity in OV-5 and 'send patrol area' message in OV-6c
2	Accident	Artillery hits special forces	Helicopter movement, landing, and deplaning of troops	Artillery fire, helicopter movement and deplaning are all nodes in OV-5
3	Deviation	Unreliable sensing	Errors in sensing, actions explicitly based on sensor data	Act of sensing not represented, but use of sensor data shown in needlines 2–5 in OV-2 and in messages 'send enemy position' and 'area clear' in OV-6c
4	Deviation	Artillery inaccurate	Position and time of artillery fire, variation in position actually hit	Not represented

Table 4.5: Example of MODAF-level Concerns

This is to be expected; the content of any model or data source will depend on its intended use, and MODAF was not explicitly designed for use in SoS hazard analysis.

Further to this, it can be noted that MODAF models that are encountered in reality are often incomplete (when compared to the full scope of the MODAF specification). Models and documentation tend to reflect the use to which they are typically put, and MODAF is not intended to be formally analysed, or converted into an executable form, so thorough study of a given MODAF document is likely to reveal a variety of omissions and inconsistencies.

The information that is provided in the source model needs to be evaluated against two kinds of needs. The first set of needs is common to all systems and models, and stems from the requirements of SoS modelling laid down in requirements M1 and M2 in Section 4.2. This can be broken down into needs for information relating to the *system* and for information relating to the *environment*. An example of the former would be the absence of an entity from an OV-6c diagram, and an example of the latter would be an absence of an identified threat in OV-1.

The second set of needs is specific to each particular SoS under analysis, and stems from the concerns that were identified in step 2. For example, the OV-2 product for the AGO doesn't identify which of the needlines will be served by radio communications, and which by other technologies.

4.6.1 Sources of Additional Information

The temptation is to provide the stock answer "ask a domain expert", but this is not always practical. Even where it is possible a modeller needs to have ways of evaluating the information they receive, and this will require them to have other sources to check it against. There are variety of sources that can be used for acquiring needed information. A precedence order for such sources, from the highest-quality to the least, is:

- Other documents relating to the specific system under analysis
- Documents relating to the *class* of system under analysis (e.g. regulations, procedures, standards)
- Domain expert opinions
- Assumptions by modellers

For each data item acquired, it is important to record the specific source (e.g. *"SoS Concept of Operations document dated 11/02/07"*), to evaluate the reliability of that source with respect to that information, and to explore the worst-case consequence of the information being wrong.

During model development, and when the model is actually used, questions will be raised about the accuracy of various model aspects. In order to adequately answer such questions, it will be necessary to draw out the sources of information that modelling decisions were based on. Failure do this may lead to questions that can't be adequately answered, reducing confidence in the model.

Making sources explicit also makes it possible to review those sources at a later date. In particular, assumptions made during the early stages of system development (because design decisions, for example, were not available) will need to be evaluated later. If those assumptions are not recorded, then engineers may struggle to perform this analysis.

When evaluating the reliability of a source, the modeller is in effect asking whether they are genuinely in a position to inform. This evaluation is not necessarily easy. The same problem, however, is present in other fields, such as requirements engineering.

Assumptions made by the modeller are given the lowest precedence in the list above, because they represent the estimates of a modeller who, in most cases, will not be a domain expert. Regardless of the modeller's relevant knowledge and experience, it is important to record the grounds on which an assumption is made. The derivation of worst-case consequence of error is also particularly important here.

The worst-case analysis is important because errors in assumptions can lead to serious errors in the predictions of the model. A pertinent example of this is the modelling of impact on the space shuttle performed prior to the Columbia accident — an assumption was made that any impacting foam fragment would strike the main wing panels, but in the event the impact was against the leading edge of the wing [90].

Whilst pessimism in the configuration of the simulation models is desirable, the derivation of worst-case effect for a particular model property can be difficult. As explained in the introduction to this thesis, the effect of a deviation of one aspect of one component system can propagate through a SoS, eventually leading to an unexpected and disproportionately large effect on the behaviour of a second component system with no direct relationship to the first. Given this, it may not be possible to derive a model property value that will result in the worst-case effect.

If it is difficult to predict the consequences of a model property, then one approach is to define a range of possible values for the missing information, and explore these as deviations (as part of the deviation process in modelling steps 6 and 7).

Despite these problems of data quality and validation, it is necessary and reasonable to supplement the source model with additional data. Results derived from a simulation model involving data that is of equivocal veracity, and indeed assumptions on highly questionable grounds, are not completely invalid; rather, they represent potentially valuable information that must be carefully evaluated before it is used.

4.6.2 Example of Additional Information

If 'information source response times may be too slow for tactical situation' had been identified as a concern in a particular SoS model, it might be necessary to know the typical time taken by an Airborne Warning and Control System (AWACS) aircraft to respond to a request to identify a radar trace. This information is unlikely to be provided in a normal MODAF model.

The first source would be to look for other documents describing the system under analysis, perhaps a high-level functional requirements document for the AWACS aircraft in its typical operating condition.

If this did not exist for the current SoS, is possible that this would be part of the documentation for a different (but similar) SoS.

If neither of these sources provided the information, it might be possible to draw upon generic standards for performance of air-traffic control or air operations information centres.

If these did not exist, it would be possible to ask a domain expert, particularly one familiar with AWACS operations (either as an AWACS crewmember or as a pilot on a client aircraft) how long such a request would typically take to be resolved.

If none of these sources were available, the modeller could simply make an assumption for the typical response time, say '10 seconds'). If there was concern that a poor choice for value might have deleterious effects on the validity of the model, it could be recorded that alternative values (such as, for example, 5 seconds and 20 seconds) needed to be evaluated as deviations of the normal case.

In any case, the modeller should record the information that was sought, the value(s) that were found, and the source from which they were taken.

4.6.3 AGO Case Study

It is clear from the description of how the AGO SoS works that the artillery will be bombarding the same set of locations that the infantry will eventually be deployed to. There is therefore the obvious safety requirement that artillery and infantry attacks are not pursued simultaneously on the same location. Examining the sequences described in OV-5 and OV-6c does not provide any particular clues as to how this is meant to be achieved.

For the purposes of the AGO case study, the assumption will be made that this interaction is achieved by providing disjoint targeting criteria for the artillery and the infantry/helicopters. Entities in the COP that are seen by the artillery as valid targets for bombardment do not appear to the infantry/helicopters as valid targets for a ground assault, and vice-versa.

The worst case consequence of this assumption is that, with the real targeting criteria *not* being disjoint, the hazard analysis would miss the potential for accidents in which the targeting criteria overlapped. Were this a real system, it would therefore be a priority to at least review this assumption with the individuals responsible for SoS tactics and rules of engagement.

4.7 Step 4 — Decompose Scenarios into Vignettes

MODAF models typically include scenarios, which are a description of a situation in which the system is involved, including mission goals, the geographical environment and the threats present. From this starting point, a wide variety of event sequences could result given different behaviour by system elements or external agencies.

Confusingly, the term 'scenario' when used in software development (and, consequently, in the Prometheus multi-agent modelling method, which will be used in Step 5) refers to a single event sequence. Padgham and Winikoff, in [6], state that *"The core of a scenario consists of a sequence of steps"*. They note that their definition is derived from the use case scenarios described by Jacobson et al. in [91].

For the process described in this thesis, the large, open-ended scenarios described by MODAF models are used to derive a set of smaller *vignettes*. [92] defines a vignette to be *"A self-contained portion of a scenario"*. In this thesis, 'vignettes' are sub-scenarios that are small enough in terms of time and geographic extent to be rendered entirely within the simulation engine.

For compatibility with the Prometheus method, a vignette can be described largely terms of a single sequence of events. This, however, represents only the 'normal' or 'sunny day' sequence. When deviations are applied to the system, or to the vignette itself, this sequence will change accordingly according to the logic of the simulation.

Each vignette that is derived should have a specific purposes in terms of exercising specific model features or exploring specific concerns. Vignettes can vary by system configuration, by environmental conditions, by the actions of agents that are not part of the controlled SoS (but

which are active in its environment), or by mission objectives.

4.7.1 Generating Vignettes

A MODAF model may already contain descriptions of a number of vignettes. Such vignettes are valuable in that they implicitly capture the concerns of the domain experts that produced the model, but they need to be carefully evaluated by the modeller to assess their adequacy with respect to the concerns identified for the system.

Vignettes can be derived by a brainstorming process. A valuable first step is to draw out some vignette elements that are relevant to each concern. Table 4.6 identifies several categories of elements. Once such a set of scenario variations has been identified, the possible combinations of those variations can be generated to produce a large number of candidate vignettes. The modeller can then select a subset of these candidates for detailed definition and eventual implementation.

Category	**Aspects**
Mission	Objectives, measures of effectiveness, intelligence available
Terrain	Effect on movement, sensing, communication and other actions
Peer/neutral entities	Allied forces and civilians (including settlements).
Threats	Number, type, objectives, capabilities, and prior intelligence of SoS configuration and capabilities.

Table 4.6: Categories of Vignette Elements

The modeller should remember that in step 7 (Section 4.10) deviations will be generated that apply to entire vignettes. It is a modelling decision, perhaps largely dependent on the simulation engine support and expertise available, whether to represent a situational variant as a specific vignette or as a deviation that can be applied to multiple vignettes.

Vignettes are described by recording the following information items:

- The state of the world at the start of the vignette
- The sequence of events that occurs during the *normal* course of the vignette
- The state of the world at the end of the vignette

Appropriate notations for describing the sequence of events include UML interaction diagrams (as used in MODAF OV-5 and OV-6c). The Prometheus method defines a notation for the expression of scenarios which might at first seem appropriate, given that the Prometheus process will later be used. This, however, is unsuitable in that it does not distinguish between the different agents; rather, it describes the overall behaviour of the SoS as if it was one amorphous

entity, eliding away the detail of how this is achieved through the actions of its constituent component systems.

4.7.2 AGO Case Study

The AGO case study defines a single scenario, which is broadly described in Section 3.8 and in more detail in Appendices A and B. Taking the concerns identified for the AGO system in Section 4.5.3, and drawing on Table 4.6 the following potential elements can be identified:

1. Airspace containing additional allied or civilian aircraft
2. Mountainous terrain that will disrupt sensing
3. Multiple simultaneous actions

Taking all combinations of the AGO vignette elements identified above provides 8 possible vignettes. As an example, we can pull out the combination ('mountainous terrain', 'multiple simultaneous actions'). This will be carried forward for the rest of the AGO study.

The start situation for the vignette can be described as:

The system units are located in a mountainous region of desert terrain. There are four UAVs, four helicopters, and three UGVs. A set of six enemy units are located within the hills and mountains in the centre of the area. Waypoints have been assigned to the UAVs that will cause them to move in a complete search pattern of the area containing the enemy units.

In this case, the sequence for the vignette corresponds entirely to the sequence for the scenario as a whole, which is shown in figure B.1 in Appendix B.

The end situation can be given as:

All component systems are still present and operational. The helicopters are now landed at positions in the central region. All enemy units have been disabled and the areas they were occupying secured on the ground.

4.8 Step 5 — Transform Domain Model to Design Model

Once the MODAF model is in place, extra information has been gathered to allow adequate representation of concerns, and vignettes have been identified, the domain model has been established. From this, the design model can be developed using the Prometheus development methodology.

Padgham [6] describes Prometheus thus: *"The Prometheus methodology defines a detailed process for specifying, designing, implementing and testing/debugging agent-oriented software."* Prometheus has been developed over a number of years, and has previously been used for developing agents for simulation models [93].

The Prometheus process was adopted for this work because MODAF models do not provide sufficient information for deriving design models (this has been discussed by Mittal in [65] — see Sections 2.6.2 and 2.6.4). The modeller must, therefore, perform additional modelling work between the domain model and the design model. This is compounded by the very variable quality and completeness of MODAF models in practice (this was noted by Zinn in [67] — see Section 2.6.2, and has also been the experience of the author). It would be possible to develop a method for improving and extending a MODAF domain model (this is the approach adopted by Mittal) but is hard to justify the considerable effort involved, given that processes such as Prometheus are already available.

One reason that Prometheus is highly appropriate for the current purpose (when compared to other multi-agent software development methods) is that it uses the Belief-Desire-Intention (BDI) model for describing the internal mechanisms of agents. In this model, which was originally introduced by Bratman in [94], agents have *desires* (goals) that they attempt to achieve by adopting *intentions* — typically, intentions are implemented by predefined *plans*, which consist of series of steps to be performed. Agents maintain *beliefs* about the world they inhabit, which inform the goals they attempt to achieve and the details of how they carry out their plans. The interaction of each agent with its environment is expressed in terms of *percepts* containing information about external events and *actions* that the agent performs.

The of use of BDI structure helps with detailed analysis of agent behaviour, as will be shown in Section 5.7.1. It has also been claimed (e.g. see McIlroy and Heinze in [95]) that BDI models, and the log output that they naturally produce, are easier for humans to comprehend than most rival models. This may make it easier for human analysts to gain insight from the study of logs or log fragments.

In their review of multi-agent modelling methods, Sudeikat et al. noted that Prometheus was a promising method for developing Belief-Desire-Intention (BDI) models [96]. It can be noted, however, that the Prometheus method excludes some aspects of BDI models that are useful for analysis, such as explicit goals held by individual agents. Agent goals, in Prometheus, are expressed only as trigger conditions on the associated plans.

By adapting the guidance provided by the Prometheus process where appropriate, the method presented in this thesis benefits from the maturity of this existing process. It is therefore possible to have additional confidence in this aspect of the process beyond the evidence presented here, and for users of the method to benefit from future developments in Prometheus.

The Prometheus process is divided into three phases which the developer is expected to perform as part of a larger iterative process. These stages, and the artefacts within them, are shown in Figure 4.3.

The mapping between top-level Prometheus artefacts and the domain model is as follows:

- *Scenarios* correspond directly to the (normal case) sequences of the vignettes.
- *System Goals* can be derived from SV-4 ('Systems Functionality Description') and also

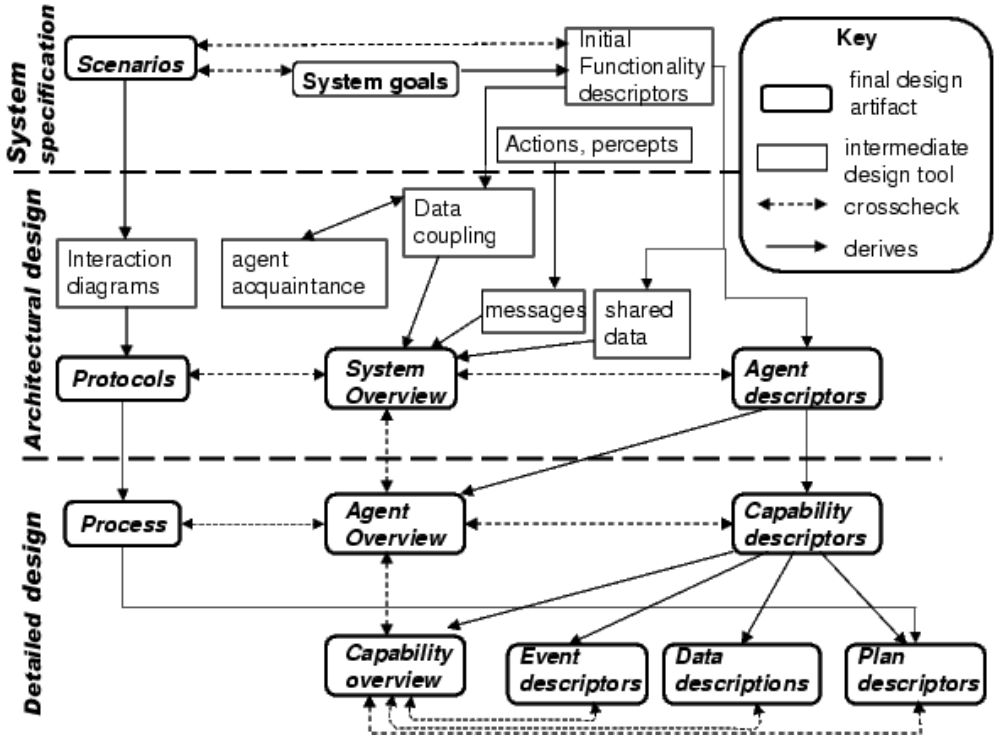

Figure 4.3: The Phases of the Prometheus Method (reproduced from [6])

from SV-5 ('Operational Activity to Systems Functionality Traceability Matrix'). To an extent, the explicit goals of the system may have to be inferred from the context given in the OV-1 and AV-1. For example, if OV-1a shows enemy platforms of various types, then the SoS is likely to have goals relating to destroying (or protecting itself from) such platforms.

- *Initial Functionality Descriptors* can be based on the agent roles implied in OV-5 and OV-6c.

- *Actions & Percepts* are more difficult to describe. In many cases, their existence must be inferred from the event sequences given in OV-5 and OV-6c; MODAF models confine themselves to only describing (directly) the interactions of the component systems in the SoS; they give little attention to the interaction of SoS component systems with the environment and any other entities that it may contain.

  For example, an OV-6c sequence diagram may show a UAV sending a message that reports the presence of a new enemy unit. If there is no earlier message received by the UAV that provides that information, then it can be inferred that the UAV detected the enemy itself. Similarly, if a component system spontaneously sends a message indicating that it has performed a particular action, then one can infer that the action is supposed to occur shortly prior to the message.

It is possible to derive some artefacts for the Architectural Design phase of Prometheus directly from the MODAF model. However, for effective use of the Prometheus process, particularly the cross-checks between the various levels (see Section 4.8.3), it is best to derive the System Specification artefacts from the MODAF and then follow the Prometheus process from there in full. This is particularly important when using the Prometheus Design Tool (described in [97]), which performs automated cross-checking and propagation of model elements between levels.

Once the artefacts of the System Specification phase have been derived, the Prometheus process can continue as described in the main Prometheus text [6].

4.8.1 Preservation of Systems of Systems Characteristics

As noted in Section 4.2.1, a SoS model must, as well as representing explicitly identified system-specific concerns, adequately represent those characteristics that are common to all SoS. When discussing methods and notations, a distinction can be made between three levels of achievement of a requirement:

1. Notation can represent the necessary information
2. Notation has explicit, helpful support for information
3. Method encourages and guides the user to adequately consider and record the information

Taking concurrency as a focus, an example of level 1 is that an unstructured text description of a software system design *can* express the concurrency in the system. At level 2, a UML sequence diagram expresses concurrency explicitly and clearly — it is the main focus of that diagram type. For level 3, Larman, presenting a tutorial description of the Unified Process in [98], states that a sequence diagram should be produced during the first 'elaboration' iteration of the development process, and that a diagram should be produced for each non-trivial scenario in each identified use case.

Table 4.7 shows how the Prometheus method and notation capture the information that is necessary to represent SoS characteristics.

4.8.2 Granularity and Level of Detail

However, additional concerns can arise when deriving the design model. As part of the Prometheus process, the set of agents that are to be implemented in the simulation must be enumerated and described. In practice, it may be possible to get this information directly from the domain model, using one agent for each MODAF operational node. This will capture the operational view of the system held by the domain experts who developed the source model. This is valuable, but it can be noted that those domain experts may not have been working with the same

Characteristic	Can represent?	Explicit support?	Guides towards?
System goals	System Goals	Yes	Yes
Multiple components	Agent Descriptors	Yes	Yes
Component SA/worldview	Agent Overview & Capability Overviews show belief sets held by individual agents	Yes	Partly – concerned with 'data', not with 'beliefs'
Component goals	Only implicitly, in the goal-seeking effects of plans etc	No	No
Component capability	Action Descriptors, Percept Descriptors, Plan Descriptors all capture aspects of what the agent is capable of.	No – capabilities are what emerges when system is run.	Partly – have to describe actions, percepts and plans.
Geographical distribution	Implicit in various places: agent, action, percept descriptors etc	No	No
Mobile systems	Action Descriptors for movement-related actions	No	No
Autonomy	Whole method assumes that agents are exhibit autonomous behaviour based on their design and the environmental stimuli encountered	Yes	Yes
Collaboration	Implicit in Protocols and in message exchange shown in the System Overview, e.g. where an agent gives orders or makes a request	No – Prometheus explicitly ignores agent-team issues	No
Comms	Protocol & Message Descriptors represent message exchange (although no representation of comms media)	Yes	Yes – process hinges on specifying message exchange at the architectural design step
Ad-hoc comms	Protocol & message descriptors, system overview diagram	No – the Prometheus artefacts only represent messages between agent *types*; networks of agent *instances* are not represented	No
Heterogeneity	Can describe entities with different behaviours	Yes	Yes – method leads towards each agent having behaviour that serves its identified purposes only

Table 4.7: Representation of SoS Characteristics in Prometheus

concerns as the safety analysts, and that this decision will effect the concerns that the simulation can support.

For example, consider the case where the domain model involves an AWACS aircraft with

multiple crewmembers. It can be noted that a breakdown in communication between two such crewmembers was held to be partly responsible for the Black Hawk friendly fire accident, as described in Section 1.1.1.

A decision to combine two (or more) component systems into one agent is potentially valuable in that it will reduce the complexity of the model, both in terms of effort and difficulty for designers and implementors and in terms of the computational requirements for adequate exploration of the state space of the model. Such compression will also reduce the time taken for any automated analysis, and probably the amount of 'noise' (data that does not contribute to analyst understanding of the SoS) in the resulting output.

For example, in a model of the Black Hawk accident, treating the AWACS aircraft as a single agent would reduce the complexity of implementing it and would reduce the number of interactions that needed to be considered in simulation and analysis. For many concerns, such as collision between the AWACS and other aircraft, or regarding vulnerability of the AWACS to enemy fire, a single agent could model the AWACS perfectly well. The fact that the AWACS 'forgot' about the Black Hawks after they registered with it could be represented as a possible failure at the agent level.

Alternatively, a decision to expand a single component system into multiple agents can expand the range of concerns that can be explored using the model, because interactions between those two agents can now be revealed as a source of hazards.

For example, if the AWACS was split into multiple agents, it would be possible to model interactions between the Enroute Air Controller and the No Fly Zone Air Controller and observe their effects on the behaviour of the SoS.

In order to decide on the final set of agents in the design model, the analyst can perform a systematic review of each entity in the domain model. For each entity, they should consider whether it could be combined with one or more of the other entities, or whether it could be subdivided into multiple agents.

4.8.3 Verifying the Design Model Against the Domain Model

Once the design model has been derived, it needs to be verified against the domain model. It could be argued that this is an unnecessary effort, given that the domain model provided the top-level artefacts of the Prometheus method, and that a well-defined process was followed. Prometheus specifies a set of cross-checks that are to be performed between the artefacts produced in its three phases. Each cross-check is a way to confirming that one particular Prometheus artefact is consistent with another.

For example, one of the cross-checks defined for the System Specification phase of Prometheus is *"Confirm that every goal is in a scenario and a role"*. This ensures that all identified goals are carried forwards into the later stages of the process. An example from the Detailed Design phase is *"Confirm that every agent produces and receives all messages that are defined for it in*

any protocol specifications". This ensures that the individual agent models are consistent with the system-level descriptions of agent interactions.

Padgham and Winikoff note in [99] that such cross-checking (particularly when automated) is effective at detecting errors made during the Prometheus process. The cross-checks, therefore, give confidence that some degree of consistency between the domain model and the final design model has been achieved.

The MODAF source model, however, also contains details equivalent to the later stages of the Prometheus process; it not only describes the goals and requirements of the system, but also describes its architecture, and in practice it will embed assumptions about lower-level system design as well. For example, interaction diagrams are part of the Architectural Design phase in Prometheus, but they are also provided by MODAF in product OV-6c. The interaction diagrams used in Prometheus need to be consistent with these models.

It can also be observed that Prometheus is intended to be a system design method; it is not specifically designed for the development of *simulations*. Consequently, the process attempts to lead the developer to develop the best design for their operational needs (as captured in the System Specification), rather than to build one that manifests behaviour equivalent to something else. It follows that Prometheus does not define any cross-checks to ensure accuracy or consistency with respect to the domain model.

This can be rectified by defining additional checks between the domain model, particularly the MODAF component, and the Prometheus design model as it develops. Table 4.8 describes some cross-checks to perform between the MODAF model and the artefacts of the Architectural Design phase. Table 4.9 describes equivalent cross-checks for the Detailed Design phase. It can be noted that the latter is rather simple, but this is to be expected as the majority of cross-checks on the Detailed Design that are defined by Prometheus as standard are against the architectural design, and are therefore still applicable here.

It is important to remember that, as noted in the discussion of requirement M5 (Section 4.2.5), inconsistency between the source model and the design model isn't necessarily wrong. Particularly in the early stages of the development process, the architecture described by the MODAF model may not, in fact, be possible, practical or sensible to implement. In effect, the MODAF model may be demonstrably wrong. In developing the design model, the modeller is moving towards an actual implementation. If there are problems inherent in the architecture they may be encountered at this stage. For example, it may be that the identified vignettes can only be implemented by introducing communications channels between agents that do not correspond to any of the needlines shown in MODAF OV-2.

When an inconsistency is encountered between the domain model and the design model, either the inconsistency needs to be rectified by modifying the design model, or the inconsistency needs to be justified by an argument as to why the MODAF model is unimplementable. This justification needs to be recorded so that it can be referred to and validated later.

Architectural Design Artefact	MODAF Product	Check
System Overview	OV-1	All MODAF operational nodes represents by agents
	OV-5	Combination of protocols and individual messages could give rise to all interactions described in OV-5
	OV-6c	Combination of protocols and individual messages could give rise to all interactions described in OV-6c
Protocols	OV-2	Protocol only exchanges data over connectivity identified in OV-2
	OV-3	Protocol only exchanges data over needlines identified in OV-3
Message Descriptors	OV-7	Message descriptors consistent with message formats described in OV-7

Table 4.8: Cross-checks Between MODAF and Architectural Design

Detailed Design Artefact	MODAF Product	Check
Agent Overview, Functionality Overview	OV-6b	Can impose a set of states on the agent consistent with the operational node states described in OV-6b

Table 4.9: Cross-checks Between MODAF and Detailed Design

4.8.4 Validating Design Model Against Reality

It was noted in the previous section that the development of a design model adds detail to the model and extrapolates from the architecture described by the MODAF model to a design that could be implemented. This may reveal flaws in the source model or in the intermediate stages. It is therefore desirable to explicitly validate the design model with respect to the world as it is understood.

The practical avenues for doing this, at this stage of model development, are limited. As the

model is not yet implemented, it output cannot be studied, nor can the running model be animated.

One approach that *is* applicable at this stage is to present the design model to domain experts for review. Sargent, in [88] calls this 'face validity'. Similarly Law, in [100], suggests that a structured walk-through of a design model, performed with a panel of domain experts, can be valuable. However, the principle means used in this thesis to ensure that the model (during model development) is adequately representative of reality is to trace the analyst's concerns through all stages of the modelling process. This is discussed in the following section.

Once the model is implemented, the analysis process described in Chapter 5 can be carried out. The potential hazards that are identified by this process can then be individually validated. The dramatically reduced scope of such validation (compared to the validation of the entire model for the purpose of hazard analysis in general) allows a wider range of techniques to be profitably employed. Such validation is discussed in Section 5.8.

4.8.5 Trace Concerns

As well as verifying the design model against the source model, the modeller must verify that the concerns identified during the domain model stage are suitably represented. As noted in Section 4.3, this tracing of concerns helps to ensure that the final implemented model addresses the concerns that were originally raised by the analyst.

For each concern, the modeller must identify the design elements that capture the information needed by the concern, and how they do so. This should be recorded for each concern.

We will return to the issue of tracing concerns for each of the later steps in the modelling process.

4.8.6 AGO Case Study

4.8.6.1 Level of Detail

The operational nodes used in the AGO MODAF model can broadly be described as 'platforms', with the exception of 'infantry' which is a group of humans (of undefined size). These are suitable for representation as agents in the design model.

In the scenario being considered, however, the infantry always travel with the helicopters. Ground movement is not considered in any detail. For the purposes of the current study, the infantry and the helicopters will therefore be combined into a single agent type (henceforth referred to as a 'helicopter'), which has the added advantage of decreasing the number of agent types that need to be modelled (and explained in retrospect). If there was a particular concern with the risks or performance implications of infantry ground movement (or stemming from the variability of same) then the infantry would be retained as a distinct agent type.

Additionally, given the limited role of the mission control entity (essentially just setting up the

mission and then waiting) it will not be modelled individually. Instead, the messages that it sends will be made part of the vignette setup (so the UAVs, for example, will start the vignette with their patrol routes already defined).

4.8.6.2 The Prometheus Model

Based on the domain model as established in the previous steps, a Prometheus model for the AGO case study was developed. Figures 4.5 through 4.7 illustrate this model graphically. A key to the notation is provided in Figure 4.4.

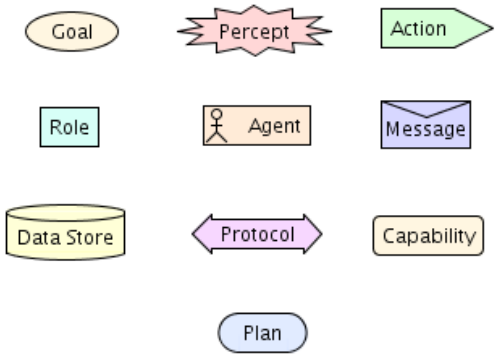

Figure 4.4: Key for the Prometheus Notation

Figure 4.5 shows the System Overview Diagram which shows the agents in the system together with their interactions with each other and with the environment. For example, notice how the only agent that does significant sensing is the UAV (because all the percepts are associated with it) and that all the agents interact via the COP (rather than by explicit messaging).

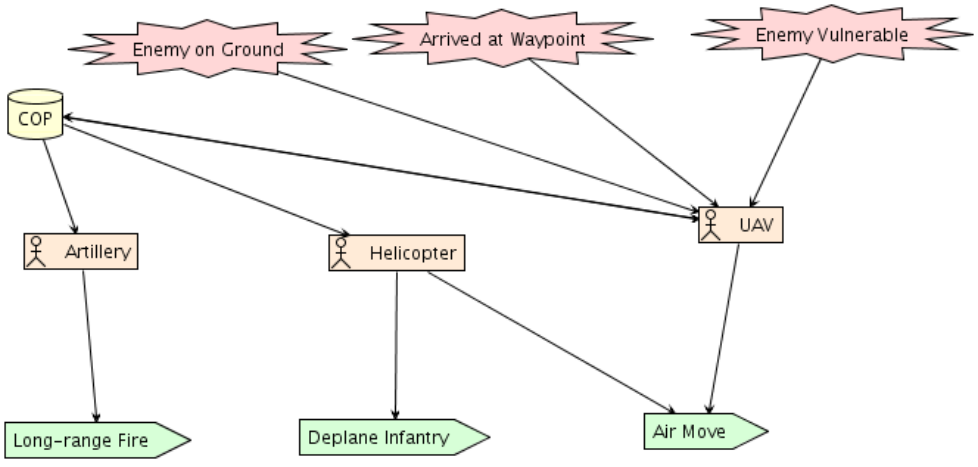

Figure 4.5: System Overview Diagram for the AGO System

The agent diagram for the helicopter agent is shown in Figure 4.6. This captures exactly how

the modelled agent interacts with its environment. It can be noted, here, that as modelled the helicopter does not directly sense the world or receive messages from other agents — its entire situational awareness and stimuli for action come via the COP. This may be considered unrealistic, but given that the details of the helicopter crew's visual perception are not the focus of our analysis, this is reasonable as a simplifying assumption.

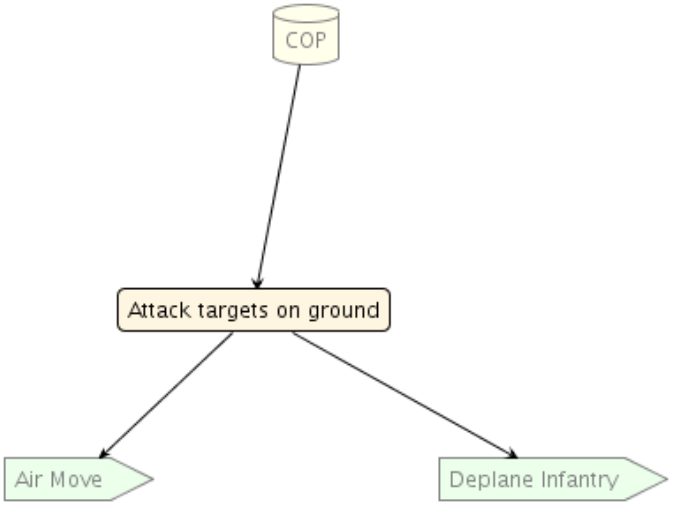

Figure 4.6: Agent Diagram for Helicopter Agent

'Attack targets on ground' from Figure 4.6 is a Prometheus *capability*, a bundle of functionality for performing a particular task. Figure 4.7 shows how the capability is made up of explicit plans, internal messages and internal data stores. The interface with the rest of the world (via actions and the COP) is also shown. This is the lowest level of diagrammatic representation in Prometheus, although further detail is captured in the textual 'descriptors' that are attached to each of the diagram elements. (Examples of descriptors can be found in Appendix C.)

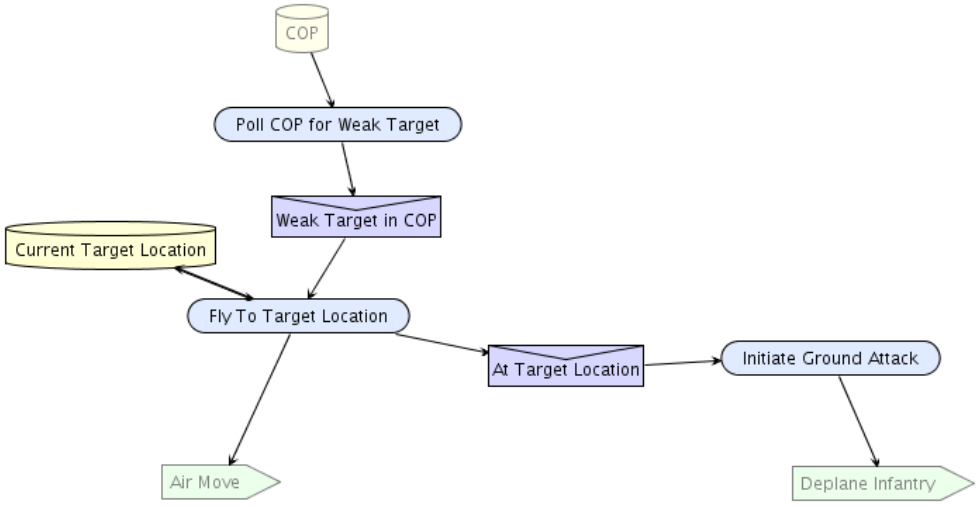

Figure 4.7: Capability Diagram for 'Attack targets on ground' Capability

It can be observed that the behaviour of the helicopter is now well defined, in an individual sense rather than purely as part of the combined SoS (for example, as shown by the OV-6c sequence diagram). This detail was not present in the domain model, so by adding it the design model has moved significantly closer to implementation.

For larger and more complex examples of the Prometheus artefacts presented above, along with their associated textual descriptors, see Appendix C — this contains a Prometheus model of the Adcock system that is discussed in Chapter 6.

4.8.6.3 Verification Against the Domain Model

During the architectural design and detailed design phases of the Prometheus process, the additional crosschecks defined in Tables 4.8 and 4.9 were applied. It is significant for this model that most of these focus on protocols and messaging, and this creates an apparent inconsistency between the MODAF model and the Prometheus model because the Prometheus model does not use any explicit messaging at all. This is partly due to the implicit representation of some component systems: the messages sent by mission control have been made part of the vignette set up, and the interactions between the helicopters and the infantry is no longer visible because they have been merge into a single agent type.

A more significant effect, however, comes from the changed representation of the COP. It was noted in Section 4.4.1 that although the fusion and dissemination of sensor data via the COP was represented in OV-6c by irregular, explicit messaging, it would be implemented in reality by a continuous ongoing process. In the Prometheus model, therefore, the COP is shown as a data store (it can be seen in Figure 4.5 and most of the other diagrams).

Of course, since the agents are physically distributed the COP eventually needs to be implemented by explicit message exchange between the agents, thereby allow each agent to maintain a local copy incorporating all the sensor data produced by the other agents. Since this functionality is common to all the agents, however, this is below the level of detail appropriate to the design model. In any case, it is only when at the stage of the operational model (step 9) that enough is known about the space, time and sensing models being used to define an appropriate representation for the COP.

Some of the crosschecks do not apply to this study. For example, the System Overview diagram is supposed to be checked against the interactions described in OV-5, and the message descriptors (if there were any) checked against the message formats in OV-7. The AGO MODAF model does not, however, identify node interactions in OV-5 and does not provide a suitable OV-7. As noted previously, the imprecise nature of the MODAF and DODAF notation frequently leads to problems of this nature. In the development of a real system it might have been that the state of OV-5 and OV-7 in the model was erroneous and could be corrected in the source model, but it is also possible that the products were deliberately produced in that way to support other stakeholders. In the latter case, the modeller would have to accommodate them.

4.8.6.4 Tracing Concerns

For the example concerns presented in Section 4.5, Table 4.10 describes how the domain-level concerns identified in Table 4.5 are expressed in the Prometheus model.

Concern	Description	Representation in Prometheus Model
1	Collision between a helicopter and a UAV	Helicopter has 'air move' action, and this is guided by a target location taken from the COP. UAV has 'air move' action.
2	Artillery hits special forces	Artillery has 'long-range fire' action, helicopter has 'air move' and 'deplane infantry' actions. Artillery and helicopters both take their targets from the COP.
3	Unreliable sensing	Helicopter and Artillery plans involve actions cued by the state of the COP, and the COP is built by the UAV based on the percepts that it receives.
4	Artillery inaccurate	Artillery has 'long-range fire' action.

Table 4.10: Example of Tracing Concerns into Prometheus-Derived Design Model

It can be observed that all the concerns are represented to some extent in the design model. Concern 4 is rather sparse at the moment, since the design model does not address the accuracy of fire at all (realistically it cannot, because the space and time assumptions will not be defined until the operational model). For deviation concerns this is to be expected, and it will improve significantly in step 6 once agent deviations have been identified.

4.9 Step 6 — Deviate Design-Level Agents

As noted in Section 3.4, a core task in hazard analysis is to enumerate possible deviations of the model. Once the normal-case representation of the system at the design level is complete, discrete deviations of the normal model can be derived. Each 'deviation' is an expression of a change to the model, such as 'agent is unable to move' or 'agent moves faster than expected'. A large part of the deviation effort is developing deviations for ('deviating') the individual agents that make up the system. (The remainder is deviating the vignettes; this is described in Section 4.10).

In the approach described in this thesis, when a deviation is applied to an agent it changes that agent *for the entire duration of the relevant simulation run*. It is possible to define deviations that cause the agent's behaviour or properties to change after the start of the run, or that affect the agent only intermittently throughout the run, but these are not considered in this thesis.

Agent deviation is performed using an approach similar to FMEA (see Section 2.4.2.3) and FHA (see Section 2.4.2.2). As in FMEA, a safety analyst works systematically over a set of components of a system, asking at each step "how could the possible deviations of this component affect the subsystem it is part of, and thereby affect the wider system?" As in FHA, a guided method is used to derive deviations (rather than relying on the results of a

previous analysis) — for each component, the analyst must ask "What deviations could possibly manifest?". The components are parts or service of agents, the sub-systems are the agents themselves, and the wider system is the whole SoS. The final step (effects at the SoS level) is deferred until the simulation is executed, but the earlier parts are performed in this modelling process step.

In the approach described here, the analyst works over distinct parts of agents and the services provided by them (analogous to FMEA 'components'), using a set of generic deviation forms to derive deviations that are specific to the agent in question. In effect, the analyst considers each of the identified 'atomic units' of the system and derives the deviations that could reasonably be expected to occur.

Unlike (manual) FMEA or FHA, no particular attempt is made at this stage to derive the effects of the deviations — that is a matter for the simulation to resolve. Of course, the modeller will need to bear in mind the identified concerns, in order to ensure that the deviations adequately explore them. They will later need to verify this as described in Section 4.11.

The generic deviation forms are derived by combining the set of generic agent parts and services with a set of guidewords. The following section provides a generic breakdown of an agent, Section 4.9.2 presents a set of guidewords that can be applied to agent parts, and Section 4.9.3 combines the parts and the guidewords to propose a set of generic deviations that are widely applicable to agents. Sections 4.9.4 through 4.9.8 then consider several additional issues.

4.9.1 Components of an Agent

Russell and Norvig, in [7], describe the basic structure of an agent as is shown in figure 4.8. It can be seen that such an agent has sensors and actuators by which it interacts with its environment and unspecified internal functionality by which sensing gives rise to action.

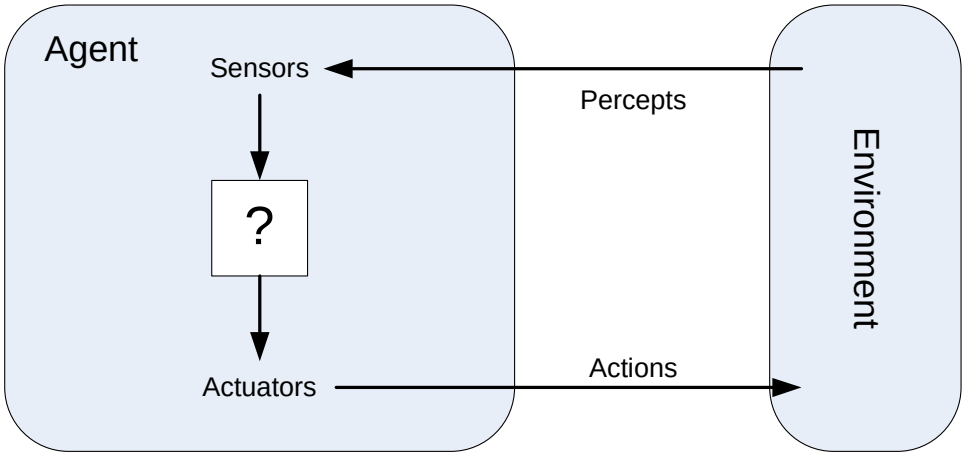

Figure 4.8: The Components of an Agent (reproduced from [7])

In the context of the current work, although the agent designs produced by the Prometheus

process have a complex internal structure, it can be observed that much of the detail of the agents behaviour is contained in the plans that are defined for it. It is therefore desirable to deviate plans specifically.

As discussed in requirement M1 (Section 4.2.1) many systems of systems accidents have involved, and are likely to involve in the future, errors relating to communications and situational awareness. It is therefore sensible to draw out these aspects of each agent for explicit, targeted deviation.

Finally, the issue of operator workload (and its machine equivalent, computational load) is frequently associated with accidents and was implicated as a factor in the Black Hawk accident described in Section 1.1.1 and the USS Vincennes accident described in Section 1.1.2. It can therefore be added as an attribute of an agent's capability.

The final breakdown of agents into parts and services that is used in this thesis is as shown below. The schematic relationships of the various parts are illustrated in Figure 4.9.

- Sensors
- Actuators
- Plans
- Communications
- Situational Awareness (SA)
- Computation / thought

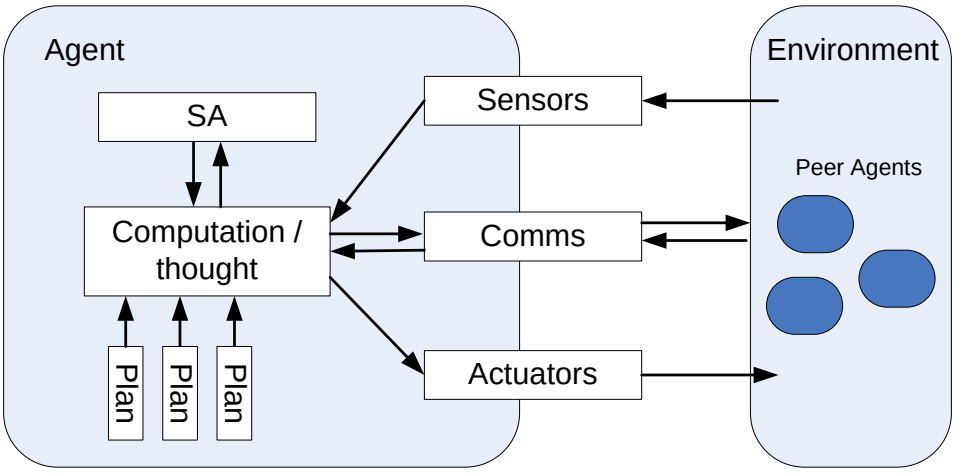

Figure 4.9: Agent Parts and Services

4.9.2 Guidewords

Many manual hazard analysis methods, such as FMEA (see Section 2.4.2.3) and HAZOP (see Section 2.4.2.4), use a set of guide words to provoke analysts to consider deviations of a system. In [101], Suokas and Kakko note that HAZOP-related methods have a strong track record of finding hazards in systems, even without automated support.

With this in mind, the approach presented in this thesis uses guide words applied to agent parts and services to derive the set of deviations that are performed in the simulation. The guidewords used are based on those used in the Software Hazard Analysis and Resolution in Design (SHARD) method, described by Pumfrey in [2], and are shown in Table 4.11.

Guideword	**Description**
Omission	An action or service is not provided, or is provided in a reduced degree
Commission	An action or service is provided when not required, or is provided in an increased magnitude
Early	An action is performed earlier than required
Late	An action is performed later than required
Incorrect	The action or service is performed incorrectly

Table 4.11: Deviation Guidewords, adapted from Pumfrey [2]

4.9.3 Generic Deviations

As noted above, a set of generic deviations have been derived from the applying the guidewords to the agent parts. These can be used to derive deviations of the agents described in the design model. Given that the parts and services adequately describe the important parts of agents, and that the guidewords provide good coverage of possible deviations of each part, it follows that the generic deviations will provide good coverage of reasonable agent deviations when applied to a specific design. The set of generic deviations is presented in Table 4.12.

There are other ways to deviate agent parts that do not arise naturally from the guidewords (and so are not shown in the table). For example, Hawkins et al. in [102] describe a method that builds on the work of Gorski and Norwicki [103] to generate mutants of software systems represented as UML statecharts. The mutations are derived by the use of guidewords on state transitions. Given a plan represented as a statechart, an approach like this can be used to automatically derive deviated versions of that plan.

Such part-specific methods are potentially valuable, however the identification of deviations for specific parts of component system models is ancillary to the central hypothesis of this thesis.

Entity Part / Service	***Guide Word***	***Generic Deviation***
Sensor	OMISSION	Total loss of sensing
		Reduction of sensor range
	COMMISSION	Duplication of contacts
		Wholly 'imaginary' contacts
		Increase of sensor range
	EARLY	N/A
	LATE	Delay in registering sensor contacts
	INCORRECT	Incorrect identification of contact side/force
		Incorrect identification of contact entity type
		Incorrect determination of contact location
Actuator	OMISSION	Total loss of function
		Reduction in magnitude of function (e.g. damage, speed, range)
		Partial loss of applicability of function
	COMMISSION	Increase in magnitude of function
		Function provided when not required
	EARLY	N/A
	LATE	Delay in performing function
	INCORRECT	Function applied to wrong target (e.g. entity, location, direction)
		General loss of precision/control (e.g. wide area, extra entities)
Plan	OMISSION	Plan step omitted
		Trigger condition not implemented
	COMMISSION	Plan step duplicated
		Extra trigger condition
	EARLY	Plan step moved earlier in sequence
	LATE	Plan step moved later in sequence
		Plan takes more thinking/processing time
	INCORRECT	Substitute entire plan with another
Communications	OMISSION	Loss of transmission
		Loss of receiving
		Entity excluded from network
	COMMISSION	Duplicate messages sent
		Duplicate messages sent later
		Entity added to network
		Additional bandwidth used
	EARLY	Message sent early
	LATE	Delay in sending messages
		Delay in receiving/processing messages
	INCORRECT	Value error in message
Situational Awareness (SA)	OMISSION	SA does not persist
	COMMISSION	Duplication of entity traces
	EARLY	Trace prematurely removed from SA
	LATE	Trace persists in SA after known to be moved/destroyed
	INCORRECT	SA coordinate system mismatched with peer entities
Computation / thinking	OMISSION	Processing tasks dropped
	COMMISSION	N/A
	EARLY	Accelerated processing
	LATE	Delay in processing (reduced processing capability)
	INCORRECT	Processing gives wrong results

Table 4.12: Generic Deviations for Agents

4.9.4 Deriving Deviation Probabilities

As will be explained in Section 5.5, a complex SoS model will provide a great many combinations of deviations, far too many for the simulation to be run for all of them. It is therefore necessary to provide a heuristic for the selection of runs. It is not sufficient to determine the combinations of deviations that are *possible*; the analyst must also determine which are *most likely to reveal hazards*. The aim must be to find the greatest number (and severity) of hazards within the finite amount of computation and analyst time that is available.

The strategy adopted in this thesis is to select those runs that are believed most likely to occur in reality. To this end, each possible run (as a combination of vignette and agent deviations) is assigned a *probability* of occurrence. Given that a particular scenario is underway, this captures the probability that the particular combination of deviations will be in place. Since the population of runs does not exist prior to the generation of probabilities, this use of probabilities is purely heuristic.

The probability of a run can therefore be computed by combining the probabilities of the agent and vignette deviations that are present. Derivation of probabilities for agent deviations is discussed in the remainder of this section; probabilities for vignette deviations are discussed in Section 4.10. It can be observed that issues of deviation independence are very salient here; these are discussed in Section 4.9.5.

Deriving probabilities is not a normal part of conventional hazard identification; it is generally performed only during risk assessment and safety analysis activities, which are later in the lifecycle. SoS hazard analysis, however, rests on the assumption that some level of component system hazard and safety analysis has already been performed. It is therefore reasonable to require probabilities of agent-level deviations at this stage.

In conventional safety analysis, the probability of a deviation is typically expressed in terms of 'failure rate per hour' or 'failure rate on demand'. It can be observed that the former can be derived from the latter if the demand rate is known. For the purposes of the approach described in this thesis, the '*deviation* rate per hour' value will be used, and the further assumption will be made that this figure is the probability of a given deviation being present during a given simulation run (by assuming a standard vignette duration of one hour).

Mechanical and electrical reliability is well understood, so it is straightforward to assign probabilities to deviations that represent simple mechanical or electrical *failures*. This has been covered by a number of texts (e.g. see Smith in [104]) and so will not be developed further here. Many potential deviations falling into the categories of 'sensors', 'actuators' and 'communications' from Figure 4.9 will fall into this category.

The HEART method [105] provides a means of estimating the likelihood of a human error, based on a combination of the type of task being performed (which provides a basic probability of an error in performance) with a set of 'error producing conditions' (EPCs) that act as multipliers on the basic rate.

For example, if a task is a *"Fairly simple task performed rapidly or given scant attention"* it has a base unreliability rate of 0.09. If it is accompanied by the EPC *"no obvious means of reversing an unintended action"* then the unreliability can be multiplied by up to 8 times. The final multiplier that was used for the EPC would have to be determined by the modeller using their own judgement of how significant the EPC would be in that precise situation.

For deviations relating to the behaviour of software, such as deviations of plans implemented by an autonomous vehicle, the assignment of probabilities to deviations is difficult. The 'probability of software' is a highly controversial topic, and naïve treatments of this have been accused of creating dangerously misleading safety analyses (see Ladkin in [106]).

One possibility is to use a figure for the probability of deviant software behaviour based on a predictive measure (of how likely it is that a given software-based system will contain an unintended deviation), however such approaches have been heavily criticised. An alternative is to derive the probability of deviation from the highest probability of correct software behaviour that can be predicted by statistical testing. This topic is developed by Littlewood and Strigini in [107], who propose a figure of 10^{-4} per hour for this. This figure can be used as an approximation of the probability of software failure for the purposes of the hazard analysis approach described here.

An alternative approach is to assume that the software components have been developed to a particular Safety Integrity Level (SIL) as specified by the standard IEC 61508 [108], and to use the associated maximum failure rate for that SIL. For example, for a SIL 3 component the probability of failure on demand should not exceed 10^{-3}. Whether such an assumption is reasonable will to a large extent depend on the development process used for the software component in question (most crucially, that it was explicitly developed in accordance with IEC 61508). It can also be noted that SILs remain the subject of controversy, particularly with respect to the question of how achievement of a given SIL can be demonstrated (see, for example, Bishop in [109] and Redmill in [110]).

For all the areas of deviation discussed above, more accurate and comprehensive methods of probability derivation, where available, will improve the quality of the hazard analysis results.

4.9.5 Issues of Deviation Independence

One probability-related issue that *does* need to be addressed, however, is that of statistical independence of deviations. A particular concern in safety is the possibility of common cause failures; if two failures are needed to cause a particular hazard, and they are each sufficiently improbable that their simultaneous occurrence appears wildly implausible, they may be safely ignored. However, this should be done only if it has been determined that there is no single underlying cause that could cause both to manifest simultaneously, and that it is itself more probable than the combination of the two.

In the current approach, the probability of a given combination of deviations is needed both for selecting runs during simulation execution and for evaluating the importance of an identified

accident rule (see Sections 5.5 and 5.6.3.6). The model must therefore capture common-cause issues.

4.9.5.1 Multideviations

One approach to doing this is to manually identify 'multideviations' which are imposed on the whole system and which 'expand' to impose individual deviations on multiple agents. It can be observed that there is some overlap here with deviation of vignettes, as described in Section 4.10, but this is discussed here because of the connection with common-cause problems.

An example of a multideviation would be the presence of effective enemy jamming that cause widespread disruption of radio communications. This might expand to one deviation for each agent that disabled its radio communication capability. The significance of this is that this multideviation would be assigned its own probability, which would likely be higher than the equivalent set of individual agent deviations (which represented, for example, electronic failures).

Multideviations can be derived by reviewing the set of agent deviations with regard to known common causes.

4.9.5.2 BETA Factors

An alternative approach can be derived from the BETA model of common-cause failure (see Smith [104]). In the original, and simplest, form of this model, the analyst assumes that a fixed proportion of all failures will be due to a common cause and adjusts the probability of combinations of failures accordingly. Developments of this include Partial BETA [111] and BETAPLUS [104], which use more sophisticated methods for deriving common-cause probabilities given the properties of the system or component that is being analysed.

Typically, such models justify their approach by reference to historical data. There is, unfortunately, no adequate source of historical or empirical data for common-cause failures at the level needed by the approach in this thesis. However, one could adopt a simple Partial BETA approach by deriving the probability of two failures having a common cause based on the relationships between the components on which they occurred. Some examples of such relationships are shown below, with example probabilities — if empirical data were available, the categories and values used could be based on this. No such data is extant, however, for the types of SoS component system with which this thesis is concerned.

Degree of Commonality	Multiplier
Different part	0.1
Different agent	0.1

Table 4.13: Multipliers for Deriving BETA Factor

To calculate the probability of a combination of deviations given these assumptions of common-causes rates, the analyst first must derive the BETA factor by multiplying together all the partial factors that apply. The contribution of any factor may be assessed by the analyst (using engineering judgement) to be anywhere between the value given in Table 4.13 and 1.0 (the latter representing the case where the factor is not present or is judged to be irrelevant).

Once the analyst has derived the BETA factor, they then multiply it by the probability for a single deviation (if the deviations being considered have diverse probabilities, these can be averaged). The result is the probability of those deviations occurring simultaneously.

That is:

$$\beta = 1 \cdot \beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_n \tag{4.1}$$

and

$$p_m = \beta \cdot p_s \tag{4.2}$$

where β is the overall BETA factor, β_n is the partial BETA for factor n, p_m is the probability of multiple deviations and p_s is the probability of a single deviation.

For example, it might have been determined for a particular agent that the probability of the deviation 'reduced optical sensor range' was 1×10^{-2} and 'increased cruising speed' was 1×10^{-3}. Without allowing for a possible common cause, the probability of a run in which both were present would be 1×10^{-5}. As these are on the same agent, however, using the Partial BETA equation with the factors given above would give a BETA factor of $1.0 \cdot 0.1 = 0.1$ from Equation 4.1 and a common-cause deviation probability of $0.1 \times \frac{10^{-2}+10^{-3}}{2} = 5.5 \times 10^{-4}$ from Equation 4.2.

Partial BETA should only be applied to combinations of deviations where there are reasonable grounds for expecting a common cause. This will require the application of engineering judgement above and beyond any historical common-cause failure data. Methods exist that structure this decision-making process, such as the Unified Partial Method [111], although these are not specifically designed for SoS. Failing to do this, and applying the Partial BETA calculations to all combinations of deviations, will result in unreasonable high probabilities of common-cause failure.

It can be observed that if the application of Partial BETA is to be automated (as in, for example, the automated generation of deviation permutations discussed in Section 5.5) then these engineering judgements will need to be made ahead of time and recorded in a machine-interpretable form.

4.9.6 Reducing the Search Space

Given a large number of deviations, it is highly desirable to reduce the set of combinations that need to be considered as this will reduce both the computation needed for any processing of the results and the analyst effort needed to interpret them (see the discussion in Section 5.1).

One approach to this is to rule out those combinations of deviations where one or more deviations is rendered irrelevant by the others. For example, there is no need to consider the combination of the deviation 'total loss of communications' with the deviation 'communications send delay'; the latter can have no effect on the outcome when the former is present.

It is also possible to rule out those runs containing deviations where the gross effect is well-known ahead of time. For example, if the events of a vignette hinges on the coordination provided by a single central command agent, then applying a deviation such as 'total loss of communication' to this agent has a predictable effect and so can be excluded from processing. However, caution should be applied when making such scoping assumptions.

When a decision is made to exclude certain deviations or combinations of deviations from consideration, it must be recorded, both in a form that can be interpreted by the simulation engine and in a human-readable form with a justification of the decision. Given that such scoping decisions can have a substantial impact on the validity of the results, they form part of the 'backing argument' for the adequacy of the analysis process, as described in Section 5.13.

4.9.7 Deviation Detection and Annunciation

In the approach described in this thesis, a deviation that is imposed on a given simulation run applies for the entire duration of that run. It can be noted that, in reality, many deviations (particularly when they are expected or easily observable failures) will be apparent to humans within the system. Once a deviation is observed by an intelligent actor in the real world, it is likely that they will take action to rectify the deviation or mitigate any adverse effects that it will have.

It is possible to build cognitive agent models that are able to respond effectively to deviations in their own capabilities, but (in the absence of general-purpose human-like cognitive agents) realistic, adequate handling of all deviations is unlikely to be practical for the approach described here.

The alternative is to perform deviations using the assumption that they remain undetected until they lead to an accident or until the end of the vignette. To some extent, this assumption can be incorporated into the probability of the deviation 'occurring' — deviations that would be relatively obvious to a human operator should have their probability of occurrence decreased commensurate with the probability that they would be detected before they could have a serious effect.

Other assumptions consistent with this are that deviations are latent until 'demand' (i.e. until they effect externally observable events in the simulation) and that the onset of the deviation is

the precise time that the vignette starts (so there is no chance of detection prior to that).

Adding the ability for a component system to have transient failures adds significant complexity to the implementation of the simulation. As such, this feature (including the incorporation of in-vignette repair/workaround probabilities) has not been implemented within this thesis.

4.9.8 Use of Existing Component System-Level Analyses

In some cases, the modeller may be able to acquire an existing set of deviations for an agent. In theory, most agents will correspond to some identifiable real-world system for which a comprehensive safety analysis has already been performed. This is usually mandated by safety-related standards and legislation; for example, systems acquired by the UK Ministry of Defence must be certified safe according to Def Stan 00-56 [112]. 00-56 requires that all hazards that a system could exhibit be identified and recorded, and this is a good source of information for imposing deviations.

It can be noted that the component systems that make up a SoS generally exist in other roles before the SoS is convened — it is rare to build a SoS that contains only new or bespoke elements.

There is a complication here, however, in that such component system-level analyses will concentrate exclusively on those deviations that can cause the component system to exhibit dangerous behaviour on their own in a context of use that is abstract and speculative, and often on situations where the component system is operating alone. It is unlikely that the assumptions of the safety engineers performing that analysis will be consistent with the actual SoS environment in which it is placed. There is, therefore, a need to explore failures and deviations that haven't necessarily been labelled as contributing to hazards.

4.9.9 AGO Case Study

Several agent deviations have been derived for the agents in the AGO model. These are presented in Table 4.14. This set is far from exhaustive — only the UAV and artillery have been deviated, and there are many other potential deviations that could be derived. It will, however, suffice for illustration. For a more extensive deviation table, relating to the Adcock system that is discussed in Chapter 6, see Appendix D.

As described in Section 4.9.4 a probability can be derived for the occurrence of each deviation. For example, if the UAV's loss of communications was attributed to a software error, a provisional probability of 10^{-4} could be assigned to represent the lowest probability that could be guaranteed by statistical testing.

The deviations presented in the table appear to be independent, but the Partial BETA technique described in Section 4.9.5 could be applied to derive an increased probability for each combination of multiple deviations on the same agent.

Agent	Part/Service	Gen. Deviation	Description
UAV	Comms	Loss of transmission	Loses ability to send out radio messages (hence, cannot update COP).
	Air obstacle sensor	Total loss of sensing	Loses ability to detect other entities in the air (hence cannot adjust course to avoid collisions).
	SA/worldview	SA coordinate system mismatched with peer entities	All entity updates to the COP are 'moved' a fixed distance from their actual location in one of the cardinal compass directions.
Artillery	Artillery fire actuator	General loss of precision / control	Effects of artillery fire are applied in a wider-than-usual area around the target location.
	SA/worldview	SA coordinate system mismatched with peer entities	All entities in the COP appear to be 'moved' a fixed distance from their actual location in one of the cardinal compass directions.

Table 4.14: Agent Deviations for the AGO Case Study

4.10 Step 7 — Deviate Design-Level Vignettes

As noted above, deviation is performed not only on the agents in the system (and, thereby, on the system itself) but on the environment that the system is placed in and the way that the system is configured within that environment.

For each discernable entity within each vignette, the analyst should consider whether it has any potential to change. In some cases, it may still be appropriate to apply the guidewords given in Section 4.9.2. A set of aspects of vignettes on which deviations can be applied is given in Table 4.15.

Aspect	Example
Weather	Heavy rain and clouds
Terrain	More and higher peaks
Peer/neutral agents	Increased civilian traffic on roads
Threat agents	Man-portable anti-aircraft weapons supplemented with vehicle-mounted systems
Mission parameters	Maximum acceptable duration of mission shortened

Table 4.15: Aspects of Vignettes that can be Deviated

It can be observed that there is often a curious reluctance to assign probabilities to future scenarios (see Ha-Duong in [113]). However, as with the agent deviations, if vignette deviations are performed without assigning probabilities, modellers can be mislead into spending large amounts of time and effort considering deviations, hazards and event sequences that in real-

ity are wildly improbable. Hastie and Dawes, in [114], note that in scenario-based reasoning decision-makers are prone to fixate on distinctive or extreme scenarios.

Therefore, as with agent deviations, probabilities must be assigned to vignette deviations so that runs containing them can be evaluated for plausibility. The normal case vignette can be considered to have a probability of 1, and indeed it is possible to have multiple 'normal' cases, in the sense that the system is expected to definitely encounter that vignette during its operational lifetime. In effect, to create such a normal case is to say that occurrence (or not) of such a situation is out of our control, and any analysis must be performed with it both occurring and not occurring. This is analogous to the use of 'normal' or 'house' events in a fault tree (see Section 5.11.2) where quantitative results must be derived with each (relevant) normal event both occurring and not occurring.

The choice of vignette deviations, and their associated probabilities, should be derived from an analysis of likely operational situations. They should also be adjusted in light of the operating policy and procedure, in that they should both test the effect of the policy (at least insofar as it refers to the system's environment) and be influenced by the kind of situations that the policy will allow to arise. Caution is important here, given the level of uncertainty over the future use of any SoS. For example, if the modeller is confident the SoS would never be deployed in circumstances corresponding to a particular hypothesised vignette, then the vignette *could* be dropped entirely. There may be a risk, however, that a situation could change after deployment so as to be equivalent to that vignette.

Adequate deviation coverage for vignettes may be expensive to achieve, particularly given that it may require the design and implementation of additional environment and agent models, but there is a long history of military operations encountering problems due to changes in conditions (one example being the problems encountered in Oman in 2001, where some British Army vehicles set on fire due to sand in their engines).

4.10.1 AGO Case Study

For the AGO case study, two variants of the vignette will be considered in which the distribution of the enemy within the area differs. This will help to insure against a hazard going undetected by the particular spatial arrangement of the targets. Since the system is expected to deal with any reasonable distribution of targets, this alternate vignette will be treated as probability 1.

4.11 Step 8 — Trace Concerns to Deviations

Once deviations have been described at the design model level, the modeller should verify that the deviations adequately explore the concerns identified at the domain model stage. For each concern, the deviations that are relevant to that concern should be identified. The analyst must then assess if the concern is adequately explored by those deviations. For any concerns that are *not* adequately explored the modeller should specify additional deviations until all the concerns

have been addressed.

The output of this step is a table describing how each concern is addressed by the deviations.

4.11.1 AGO Case Study

Table 4.16 gives a justification of why the Anti-Guerrilla system deviations described in Section 4.9.9 cover the concerns identified in Section 4.5. For the purposes of this case study, these deviations are adequate.

Concern	Description	Representation in Deviations
1	Collision between a helicopter and a UAV	UAVs can lose ability to sense other aircraft, and their coordinate system can be changed (so their movement patterns vary from expectation)
2	Artillery hits special forces	Artillery can be inaccurate, and can have its coordinate system skewed so that its aiming for the wrong location
3	Unreliable sensing	UAVs can lose sensing completely, and have their coordinate system skewed so that they contribute traces to the COP in the wrong locations
4	Artillery inaccurate	Artillery can be simply inaccurate, can have its coordinate system skewed (so that its aiming for the wrong location in any case), and the UAV can have its coordinate system skewed so that it is contributing targets to the COP in the wrong location

Table 4.16: Example of Tracing Concerns to Deviations

4.12 Step 9 — Transform Design to Operational Model

Once the design model is complete, has had its associated deviations defined, and has been evaluated against the source model and against the identified concerns, the modeller can move from design to implementation. Drogoul, however, draws out an important interim step, that being the derivation of the *operational model*, whereby the design model is fitted into the constraints of the implementation language and tools [5].

By identifying safety concerns at the domain level, and tracing them through their representation in the design model, the modeller has reduced the problem of ensuring adequate representations of concerns in the implementation to a general model refinement problem. This part of the process, therefore, is a much more general problem than that which falls within the scope of this thesis, and so will not be covered in great detail. However, some issues that are specific to the current process will be discussed here.

4.12.1 Choice of Implementation Platform

The choice of implementation platform is likely to have significant impact on the ease of implementation, and the potential for an implementation that is faithful to the design. For example, the main Prometheus text [6] proposes that the JACK agent development platform [115] be used for implementation of Prometheus models. The Prometheus text notes that, by comparison, implementing a multi-agent system design using only conventional object-oriented languages and tools (as distinct from using a purpose-built agent platform) leads to implementations that are *"awkward and difficult to maintain"* [6].

Sudeikat et al., in [96], evaluate the compatability of the Jadex agent platform [116] for use with the Prometheus method. They note that Prometheus is broadly compatible with Jadex, but that certain assumptions differ, for example the format of messages and the lack (in Prometheus) of explicit individual agent goals.

Drogoul's concept of an operational model is mainly influenced by the social sciences modelling field in which she is active. The situation there is that space and time representations, for example, are defined uniquely for each model, based on its particular needs. In mainstream military modelling, by contrast, there is widespread use of general-purpose simulation engines, often provided on a commercial basis (or developed centrally by a government for use throughout its forces). An example of the former is FLAMES [117], and of the latter the OneSAF Objective System [118] developed by the US DoD. Such platforms invariably make significant assumptions about how time and space will be modelled, and often go further by providing existing models of vehicles, weapons, sensors and terrain.

In light of the above, the concept of 'operational model' used here can be considered to take into account such concerns. If the features offered by military simulation platforms are to be leveraged, the operational model will have to take into account the assumptions that they make.

In many situations, the choice of implementation platform will be largely dictated by availability of tool licences and skill sets. When there are multiple plausible options for the choice of platform, the platform chosen should be that which provides as simple and transparent a mapping as possible from the assumptions of the design model to the assumptions of the platform.

4.12.2 Elaborating the Environment Model

Requirement M2 (see Section 4.2.2) requires that the environment in which the simulated SoS is embedded be realistic and adequately diverse. The choice of implementation platform will, however, heavily constrain the types of environment that can be simulated, and full elaboration of the environment must be delayed until the operational model is defined.

There is no extant guidance on this aspect of SoS modelling — most of it (including the published DODAF and MODAF guidelines) concentrates on the system itself to the exclusion of its context. This is understandable given that the SoS will need to operate in many environments and contexts. However, a thorough consideration of possible environments is necessary for

adequate hazard analysis to be performed. By combining the environmental factors mentioned in several relevant sources, however, a checklist of important environmental factors can be derived. Any given model will not necessarily incorporate all possible aspects of the environment, but the following provides stimulus to consider their relevance to a given SoS model. As with other modelling decisions, the core SoS characteristics and the identified model-specific concerns should be the guide.

For example, the MODAF technical handbook [64] notes that OV-1a (the high-level operational concept graphic) covers "missions" and "geographical distribution of assets". Studying the example graphic presented in the handbook shows that the creator was concerned about elevation of terrain, and about line of sight blocked by earth or trees. The enemy vehicles are distinguished by type ('tank', 'wheeled APC' and 'dismounted infantry'), and are represented purely as targets.

The handbook includes the following relevant entries in the OV-1a metamodel:

Environment *"A general specification of the surroundings / scenario in which an operation may take place. Examples of environment would be; 'desert', 'arctic', 'at sea' etc."*

Location *"A location anywhere on the earth"* — the text later suggests that this might be represented by GPS coordinates

The MODAF model presented in [119], which will be used for the case study in Section 6, presents several candidates for each of several aspects, such as 'terrain' and 'weather conditions'.

Historical accidents also provide a source of significant environmental factors. For example, the Black Hawk example presented in Section 1.1.1 implicated the effects of mountainous terrain on radar visibility in the occurrence of the accident, and Regan [15] presents numerous examples of unexpected friendly presence leading to fratricide.

Several important environment aspects can therefore be identified:

Terrain The environment model must, as a necessity, contain a map of the terrain within the simulated area. This can, if thought necessary, be elaborated by including a representation of elevation, and of ground cover (foliage, rocks and buildings).

Weather Weather can be applied in a global form (for example, fog throughout the simulated area) or as local effects of discrete size (for example, clouds at specific positions with specific size). For either form, there is potential for effect on communications, sensing (particularly visual or visible-light), on ground mobility, and on equipment failure rates. Some component systems (particularly aircraft) may not be able to operate in some weather conditions, or enter certain weather features.

Day and Night Day and night effects are similar to global weather in that they have substantial effects on visible-light sensing.

Threats Enemy threats can be described by their number, type and location, and potentially (in more sophisticated models) by their goals or operating procedures. The use of simple reactive behaviour may be satisfactory (for threats who exist solely to fire from a fixed position) but more developed cognitive models may allow additional vulnerabilities to be identified, although this will make the results of the simulation runs more difficult to interpret (as the enemy behaviour will need to be understood in each case).

Consideration of a range of threat types (such as ground, air, special forces and electronic warfare) will provide more complete analysis, again at a cost of complexity.

Non-threat Entities Other than the component systems of the SoS and its enemies, the environment may contain friendly, allied force or neutral entities. At the simplest level, these provide potential for additional accidents. They may also disrupt expected operating patterns (such as flight paths or patrol routes). More sophisticated considerations include differences in technology (such as lack of IFF or collision-avoidance technologies).

Communications Environment The communications environment available to the SoS consists primarily of the range of communication channels available (such as telecommunications networks, radio, and satellites) and the attendant limitations on those channels (most notably bandwidth). It can be further elaborated by consideration of intermittent or unpredictable effects such as variable use of bandwidth by allies, and attempts at jamming.

4.12.3 Tracing Concerns to the Operational Model

In theory, the transition from design model to operational model is fairly straightforward. If the design is implemented correctly then the concerns will be adequately captured in the implementation. It is still necessary however, to identify and resolve any potential mismatches between the design and operational models that could threaten the expression of the concerns.

For example, if an identified concern relates to performance in mountainous terrain, this will be compromised by an implementation platform that only supports flat terrain that affects movement rates but not visibility or communication.

4.12.4 AGO Case Study

As noted in Section 2.6.3, simulation platforms that provide only simple cellular automata modelling are insufficient for adequate SoS modelling. Such models cannot capture the explicit procedures which are defined to guide SoS component system behaviour. Simulation engines such as SEAS (see Sections 2.5.5 and 2.6.5) are more flexible, but are either expensive or of

restricted availability (SEAS, for example, is only available to the US Government, its armed forces and contractors). It was therefore necessary to develop a custom simulation engine for research into SoS safety.

The Sim8 simulation engine was originally developed by the author for the work described in the DARP project (see, for example, Alexander in [120]). It provides a discrete-time, discrete-space simulation of an outdoor environment which contains a set of mobile agents. It provides the facility to build agent controllers by composing multiple distinct 'behaviours', and support for the management of agent deviations. As all agent properties and behaviours are implemented in the general-purpose Java programming language, the engine is flexible enough to express a wide variety of component systems and scenarios.

The Sim8 engine provides a simple simulation of radio communications, including a limited form of bandwidth restriction, and can detect collisions between air vehicles. The coarse-grained space and time scales used provide a measure of incident detection in that the system will indicate a collision when, in a finer-grained model, it might have been a near miss.

Sim8 uses the MASON simulation framework (described by Luke et al. in [121]), which provides basic scheduling, a range of simple space models, and simple visualisation. The additional functionality provided by Sim8 is implemented as around 8000 lines of Java source code. The representation of the two case studies presented in this thesis involved an additional 5000 lines.

Sim8 was used to implement the AGO case study. A mapping of the AGO modelling concerns to the Sim8 platform is shown in Table 4.17.

Concern	Description	Representation in Operational Model
1	Collision between a helicopter and a UAV	Sim8 tracks agent movement (according to agent-specific movement rules) and detects collisions between aircraft.
2	Artillery hits special forces	Sim8 allows indirect fire to be modelled, where all agents near to the target location are hit. Agents can be destroyed by battle damage.
3	Unreliable sensing	UAVs can be implemented so as not to sense when the appropriate deviation is applied. UAVs can maintain internal coordinates that are skewed relative to the external world coordinates.
4	Artillery inaccurate	Artillery can use internal coordinates that are mis-mapped to external world coordinates. Artillery fire can be (when deviation is applied) distributed over a larger-than-normal area.

Table 4.17: Example of Tracing Concerns to Operational Model

The Sim8 platform provides a standard mechanism for listing the deviations applicable to an agent, and for applying deviations to an agent in a particular run. It does not, however, provide standard ways to manipulate agent behaviour in response to particular kinds of deviation. For

example, inaccurate artillery fire must be specifically implemented such that when the deviation is applied, the area of effect is increased. This is not a fundamental problem (since the entire set of deviations is known at model implementation time and can so be implemented), but there is a future opportunity here to reduce model development effort by providing standard implementations of particular failures (such as, in this case, inaccurate weapons).

4.13 Step 10 — Implement Operational Model

If the operational model is adequate for the concerns and the other details in the design model, it can be implemented. In theory, if the operational model is adequately identified, this transformation is straightforward and indeed mechanical. The operational model should have captured the representation of the domain agents as *computational agents*, and this provides the specification and (at least part of) the design that is to be implemented.

Although errors in implementation of software remain a problem, the effectiveness of existing development methods means that, historically, the stages between specification and implementation have not been where the majority of errors were introduced. For example, one review [122] attributes only 14.7% of serious system failures to *'inadequate design and implementation'*.

As with development of the operational model, most of the details of an implementation approach are therefore outside the scope of this thesis. Some specific considerations, however, are identified below.

4.13.1 Tracing Concerns

One final step in tracing the top-level concerns is necessary. If the implementation is adequately representative of the operational model, then it will address the concerns *in terms of how it behaves internally*. The issue that remains is that, regardless of precisely how it is implemented, it must address the concerns *in terms of model output*. For the most part this involves logging the events and states that occur within the simulation. For example, if the effects of limited bandwidth were identified as a concern then it must not only be possible for inter-agent messages to be dropped or delayed, but it must be possible to determine from the simulation output when this has occurred.

The greater part of this logging follows obviously and automatically from the use of BDI agents. BDI engine events such as changes in beliefs, the activation of goals and the selection of plans to achieve those goals can be automatically output by the simulation engine. Support for such output is fairly standard in BDI reasoning engines and can easily be implemented.

In addition, considerations introduced in the operational model, such as space and time representation, suggest a variety of other logging needs such as entity movement and the passage of time. The analysis techniques used will also introduce particular logging requirements of their

own; see Section 5.7.1.3 for discussion of the specific requirements of agent tracing tools.

4.13.2 Verifying the Implementation Against Domain Model

The implemented form of an imperative computer program is likely to be too complex to efficiently cross-check with the domain model. This is likely to be true even if the program is expressed in a dedicated agent programming language such as Jason [123], which allows the implementation to be expressed in terms that are very close to the design model.

It is, however, possible to compare the results of the vignettes that occur in the implemented simulation model with the event sequences that were originally used to describe those vignettes (see Section 4.7). This will only be possible for the normal case of each vignette, because the domain model is not expected to contain sequences that take account of deviations.

As with previous cross-checks, if the results of a vignette in the simulation are inconsistent with the results described in the domain model, then the difference must be explained and either rectified or justified. In some cases, the difference will be unavoidable (due to practicalities of implementation) or insignificant (given the concerns).

For example, a particular pseudo-BDI implementation might log the activation of a plan both when it is initially activated and when it resumes action after waiting for some event. This would be undesirable but since it is only changing the log (not the events that 'occur' in the simulation world) it can be worked around. Similarly, an operational model that used a discretized space model might produce a 'move action' log entry once for each sector moved. Again, this does not change the events in the simulated world and so can be ignored, or resolved by post-processing the log to combine multiple moves into a single action.

4.13.3 Implementation of Deviations

The implementation of deviations is not trivial, and each deviation added increases the complexity of the implementation. To compensate for this, the modeller can prioritise the implementation of deviations, starting with those that they believe are most likely to result in hazards. Generally, these should be deviations that follow directly from the identified concerns.

4.13.4 AGO Case Study

The AGO case study was implemented in Sim8, with the vignette identified in Section 4.7.2 and the deviations identified in Sections 4.9.9 and 4.10.1. The unique aspects of the AGO agents were defined by approximately 3000 lines of Java code. Logging output was produced by recording the belief, goal and plan invocations included in or implied by the Prometheus model, along with all specific agent actions (such as movement or attacks).

As noted above, the concerns identified for the model need to be mapped to elements of the logging output produced by the implementation, thereby ensuring that the concerns can be

Concern	Description	Representation in Implementation
1	Collision between a helicopter and a UAV	Collisions between aircraft. Helicopter and UAV movement. Beliefs, goals and plans that will lead to movement.
2	Artillery hits special forces	Movement (and hence location) of helicopters. Beliefs, goals and plans that lead to artillery fire. Artillery fire actions. Destruction of helicopters (and hence the troops they carry).
3	Unreliable sensing	UAV sensing actions. UAV belief (with confidence levels) of presence and strength of enemies at a location.
4	Artillery inaccurate	Beliefs, goals and plans record intended target coordinates; fire actions and damage/destruction events record actual coordinates.

Table 4.18: Example of Tracing Concerns to Log Output of Implementation

investigated using the logs. A mapping of the concerns from Section 4.5.3 to log elements in the AGO implementation is shown in Table 4.18.

The implemented AGO simulation model was then executed for the normal case of the vignette, producing a log that described the events that occurred. This log can be compared to the sequence of events described in the sequence diagram for the vignette (see Section 4.7)).

An excerpt from the log for the normal-case run of the AGO scenario is presented in Figure 4.10. Since the sequence diagram for the vignette only includes a single instance of each agent type, the log has been filtered to include only a single instance from the several that are modelled in the simulation. In addition, many repetitive log entries (such as the movement actions of the helicopter) have been removed for clarity.

The format of each entry is *"type,agent,details,time,run number"*. For example *"event,enemy4,weak,21,0"* shows that there was an *event* at time step *21* in run *0* where *enemy4* became *weak*.

It can be observed that the log maps across to the sequence diagram shown in Section 4.7.2:

- At time step 0, the UAV believes that it has a waypoint to go to — this corresponds to the 'send patrol area' message shown in the sequence diagram.
- The UAV then patrols the area (as shown by the succession of 'next waypoint' beliefs). This corresponds to the 'Patrol Area' self-message.
- A time steps 18-20, the UAV scans an enemy position and propagates it (with increasing confidence) into the COP.
- Via the COP this belief is propagated to the artillery ('gun1') at time 19. The artillery then activates a goal and a plan that cause it to bombard the enemy with artillery fire

```
belief,uav2,next waypoint is at (22;11),0,0
message,uav2,suspect strong enemy at location (21;19)=4,18,0
message,uav2,suspect strong enemy at location (21;19)=8,19,0
message,uav2,suspect strong enemy at location (21;19)=12,20,0
belief,gun1,strong target at (21;19),21,0
goal,gun1,supress strong enemy at (21;19),21,0
plan,gun1,artillery bombardment at (21;19),21,0
event,enemy4,weak,21,0
belief,uav2,next waypoint is at (31;11),27,0
belief,uav2,next waypoint is at (31;23),36,0
message,uav2,suspect strong enemy at location (32;16)=3,40,0
message,uav2,suspect strong enemy at location (32;16)=6,41,0
message,uav2,suspect strong enemy at location (32;16)=9,42,0
belief,uav2,next waypoint is at (22;23),48,0
message,uav4,suspect weak enemy at location (21;19)=4,49,0
belief,gun1,weakened target at (21;19),49,0
belief,uav2,next waypoint is at (22;11),57,0
message,uav2,suspect weak enemy at location (21;19)=8,60,0
message,uav2,suspect strong enemy at location (21;19)=4,60,0
message,uav2,suspect weak enemy at location (21;19)=12,61,0
message,uav2,suspect strong enemy at location (21;19)=0,61,0
belief,heli1,weak target at (21;19),61,0
goal,heli1,neutralise position at (21;19),61,0
plan,heli1,ground attack at (21;19),61,0
action,heli1,move (-1;-1) towards (58;37),61,0
action,heli1,move ...
message,uav2,suspect weak enemy at location (21;19)=16,62,0
belief,uav2,next waypoint is at (31;11),69,0
belief,uav2,next waypoint is at (31;23),78,0
message,uav2,suspect strong enemy at location (32;16)=12,82,0
message,uav2,suspect strong enemy at location (32;16)=15,83,0
message,uav2,suspect strong enemy at location (32;16)=18,84,0
belief,uav2,next waypoint is at (22;23),90,0
action,heli1,move ...
action,heli1,move (-1;0) towards (21;19),98,0
belief,uav2,next waypoint is at (22;11),99,0
action,heli1,land at (21;19),99,0
event,enemy4,destroyed by heli1 at (21;19),100,0
```

Figure 4.10: Filtered Version of the AGO Log Output

— this autonomous behaviour replaces the 'suppress enemy' order that the sequence diagram shows it receiving.

- At time 49, a second UAV scans the enemy position and determines that that is now in a weakened state. It propagates this to the COP (with an initially low confidence).
- Immediately afterwards, the gun realises via the COP that its current target has become weak, and ceases fire.
- At time 61, the COP indicates sufficient confidence that the enemy is weak that the helicopter believes it, and it forms a goal and a plan to deal with the enemy. This corresponds to the 'dispatch to enemy' message shown in the sequence diagram.
- At time 99, the helicopter lands at the enemy location. This correspond to the 'deplane' message.
- At time 100, the enemy is destroyed.

There are aspects in which the log differs from the sequence diagram. For example, at time 49 the gun ceases fire. This is not shown in the sequence diagram, but is obviously necessary and the discrepancy is therefore acceptable.

More seriously, the sequence diagram shows the infantry being mobilised (and hence the helicopter being dispatched to the enemy location) before the guns open fire, whereas the log shows the helicopter starting to move at time 61, which is *after* the guns have *ceased* fire. The original sequence diagram (which was taken directly from the MODAF model provided by Despotou in [89]) presents a situation in which the guns are firing as the helicopters are moving, while in the simulation they move at disjoint times. The simulation model, therefore, appears to present a lower-risk situation (although it can be noted that this is consistent with the decision made earlier to assume that the targeting criteria for the artillery and infantry are disjoint; see Section 4.6.3).

There is no obvious solution to this second discrepancy. It *is* of concern, because it appears to affect the safety risk extant in the modelled situation. In a real situation it would need to be addressed before preceding further.

4.14 Summary

This chapter summarised the requirements for SoS simulation models, and presented a modelling approach that produces models that meet those requirements. The process was illustrated at each step using the ongoing case study that was introduced in Chapter 3. Modellers following the defined processed should be able to have some confidence that their implemented simulation model will be an accurate representation of the SoS that is to be analysed (given the specific safety concerns that were identified for the SoS), and therefore that any hazards that are found using it will prove to be plausible.

The following chapter shows how an analyst can use the defined behaviour of the models, and the sets of deviations that have been derived for them, to discover unknown SoS hazards that are present in the system. Guidance is then given on reviewing these hazards (for questions of validity and importance) and on presenting the knowledge gained by the analysis process.

Chapter 5

Analysing the System

As described in Chapter 3, analysing a system involves performing multiple runs with deviations and extracting patterns that emerge in the resulting data. Chapter 4 described how to identify the possible deviations that are of interest in the model. This chapter deals with the exploration of the space created by those deviations, and shows how to interpret the results.

5.1 The Problem of Simulation Output Analysis

Given a simulation model of the type described in Chapter 4, a safety engineer must analyse the model in order to find hazards and to understand something about them. In the current context, this involves choosing simulation runs (in terms of combinations of deviations) and deriving knowledge from the results.

At this point, there are three key problems to be tackled: the size of the parameter space, the difficulty of understanding the results, and questions about the validity of the results.

5.1.1 Problem 1 — Size of the Parameter Space

Even with a modest number of deviations for each agent, the number of possible combinations of these is extremely large (see Hoeber in [87] for some discussion of this in relation to simulation). Indeed, as the number of deviations grows (for example, through domain expert contributions as the simulation model improves through multiple iterations) this problem becomes progressively harder. It is therefore not obvious how to achieve adequate coverage of the parameter space, while neither spending an unreasonable amount of time performing runs that are of no interest or failing to perform runs that display plausible hazards.

There is a large literature on experimental designs for optimal exploration of parameters; a general survey is provided by Montgomery in [124]. Several techniques relevant to multi-agent simulations are discussed by Dewar in [56]; they include fractional factorial designs, main-effects screening, group screening, and random perturbations.

All of the techniques that Dewar identifies work on the assumption that the effect of each parameter on the safety-critical behaviour of the system is independent of the effect of the other parameters, that there is a monotonic relationship between the number of deviations imposed and the level of safety risk, and that small variations in parameters cause small changes in the output. Given the past examples of SoS accidents, and the characteristics of SoS identified in Chapters 1 and 2, the validity of these assumptions is difficult to assert for the class of SoS being addressed in this thesis. For example, it is entirely possible for one deviation to cancel out the hazardous effects of another.

Similarly, optimisation techniques (see Gray et al. in [125] for a survey) are unsuitable because they are concerned with finding a single set of parameters that gives the optimal value of a particular utility function, not with exploring all the possible ways of achieving particular results. It is possible to repeat an optimisation for each of several utility functions, but it is unclear how one would specify such functions so as to reveal all hazards, given that little is known about the set of hazards in a system ahead of time. Additionally, optimisation techniques rely on the landscape of the response to parameters having a particular general topology. The landscape generated by searching for accidents in a realistic simulation is likely to be very sparse — a mixture of large flat expanses (no accidents) with small clusters of extreme results (accidents). Most optimisation algorithms will perform poorly on such a landscape.

5.1.2 Problem 2 — Difficulty of Understanding the Results

Section 4.13.1 discussed how to select information for inclusion in logging output, and an example of the kind of log that might be produced is shown in Figure 5.1. The output that results from the reasonable exploration of a realistically complex model will be huge — thousands or millions of such run logs, each potentially containing tens of thousands of entries. It is grossly unrealistic to expect a human analyst to study such logs manually, let alone understand them. As noted in Section 2.5 this is a common problem with automated analysis tools.

In most of the sciences and engineering, it is common to use statistical methods to discover relationships between parameters and output measures. Examples of such in safety-related simulation can be seen in Blom [54] and Johnson [60]. Such techniques are not, however, particularly appropriate to the current work. As noted in [126], the frequently non-linear behaviour of complex multi-agent systems, combined with the contingency of complex system behaviour patterns (the potential for a small change at one point in time to have a large effect on the system's later behaviour — see Edmonds in [127]), means that such measures often reveal little about the system.

It can also be noted that conventional statistical models are designed for relating continuous-valued parameters to continuous-valued results, whereas the work described here is concerned with relating discrete-valued (boolean-valued) parameters to discrete-valued outputs.

More crucially, all of these techniques are concerned with producing statistical data, while in the hazard analysis approach described in this thesis we are concerned with finding all routes

```
finished_run(4).

has_entity(uav1,4).
has_entity(uav2,4).
...
has_entity(enemy1,4).
...

has_dev(fusiongridnorthwestskew,uav2,4).
...

belief,uav3,next waypoint is at (30;12),0,4
goal,uav3,travel to (30;12),0,4
plan,uav3,fly to (30;12),0,4
move(uav3,8,37),0,4
action,uav3,move (1;-1) towards (8;37),0,4
belief,uav4,next waypoint is at (23;20),0,4
...
event,heli2,destroyed by enemy6 at (34;18),129,4
...

end_time(500,4).
casualty_cost(64.0,4).
material_cost(4.0,4).
incidents(0.0,4).
```

Figure 5.1: An Example of a Log File for a Single Run

by which accidents can occur.

It can be seen that these first two problems interact; if the coverage of the parameter space is achieved in an inefficient manner, the volume of output needed to adequately cover it will be increased.

5.1.3 Problem 3 — Questions About the Validity of the Results

The third problem is that, even when the model has been adequately explored and methods have been found to make the huge volume of output comprehensible to the analyst, it is not immediately obvious whether these results are valid, artefacts of the analysis process, or artefacts of the model itself.

5.2 Requirements for an Analysis Method

With the above problems in mind, this section sets down the requirements that were established for the analysis of simulation models as presented in this thesis.

5.2.1 Requirement A1 — Analysis must selectively explore the space provided by deviations

Requirement

The analysis technique must select simulation runs to be performed so as achieve adequate coverage of the parameter space of the simulation while exploring only a small proportion of that space.

Runs must be selected so as to allow interactions between different deviations to become apparent. The selection mechanism must explore cases where the overall safety risk increases non-monotonically with respect to the number of deviations imposed, and must remain viable in the event that small changes to the deviation set (e.g. addition or removal of a single deviation) have a radical effect on the output.

Rationale

Exhaustive exploration of the parameter space demarcated by all agent and vignette deviations would be highly desirable, in that all behaviour paths that were implemented by the simulation model would then be revealed. If the set of vignettes and deviations was considered to provide an adequate description of the real situations that the system would encounter, then the analyst could easily argue that their analysis was complete. As noted in Section 5.1, such exhaustive exploration is intractable even for simple simulations with modest numbers of inputs. There is therefore a need for a more selective exploration of the parameter space.

5.2.2 Requirement A2 — Analysis must find all paths to accidents

Requirement

Once the simulation has been run for the parts of the parameter space that are to be explored, the analysis process needs to identify all the paths by which accidents have occurred. Each such path represents either a new hazard or a new cause of a known hazard. Merely identifying all the possible results (accidents and variations of accidents) is insufficient. Likewise, simply identifying the overall system risk that the system configuration presents is not enough. The definition of hazard analysis (as given in Section 2.4) requires that all hazards in the system be identified.

The techniques applied must be robust to complex non-linear interactions between model parameters and to the highly contingent nature of agent behaviour.

Rationale

In order to render the system safe, it is necessary to change the design or operational policy such that all hazards are prevented or adequately mitigated. The analyst must, therefore, identify all of the hazards that *can* be revealed given the knowledge contained within the SoS models. As noted in Section 5.1, the nature of multiagent simulation is that its behaviour is complex and difficult to analyse by conventional means. It has been observed, for example by Richardson [126] that complex multiagent systems may exhibit a wide range of distinct behavioural patterns, switching between them in response to apparently small changes in the model or the stimuli presented. There are therefore likely to be many routes by which the system could cause a particular accident.

5.2.3 Requirement A3 — Analysis must lead to results being validated

Requirement

Any hazard rules or explanations that have been derived during analysis must be validated, and the analysis approach must provide clear guidelines for doing this.

The results must, first, be validated against the model — the question is "Is this a real hazard, or is it just an artefact of the analysis technique?".

Those results that pass that test must then be validated against reality — the question then becomes "Does this hazard manifest in the real system?".

Rationale

Automated techniques used in analysis of simulation results may products some results that are artefacts of the technique rather than representative of hazards that really appear in the model.

Although the model will already have been validated in a general sense (see requirement M5 in Section 4.2.5), the inherent difficulty of this undertaking will have limited the completeness of validation that was achieved. Specific statements about cause and effect can be validated much more thoroughly.

5.2.4 Requirement A4 — Analysis must present results in human-comprehensible form

Requirement

When an automated analysis technique identifies a hazard, it must present it in terms that the analyst can understand. A black-box predictor of accidents is not sufficient.

Rationale

The analyst cannot rely on the simulation being entirely accurate. The system cannot, therefore, simply *tell* the analyst what hazards will be exhibited in the real system. Instead, it must identify things that appear to be hazards and make a case for them (see requirement A3, Section 5.2.3). Such a process compensates for the limited fidelity of the model by 'going part of the way' and then handing over to the analyst (see Zhao and Venkatasubramanian in [128]). If the analyst cannot understand the description they are given, however, it is of little use to them. They will not be able to validate the hazard, or propose a solution.

Fox and Das, in [129], note that a similar situation holds with regards to expert systems used for medical decision making; it is crucial that the decisions proposed by the computer are supported by an argument or rationale so that doctors can challenge them.

5.2.5 Requirement A5 — Analysis must present results in terms of SoS mechanisms

Requirement

We can regard analysis of the observable outcomes of the entire SoS as being an *external* analysis. However, such an analysis cannot explain how the behaviours of the constituent component systems give rise to these outcomes. For this, we require an *internal* analysis in terms of these component system behaviours.

It is important that the results of such an analysis are above the level of the 'micro-knowledge' (see Section 4.1.2) that the modeller started with — there is no value in merely rediscovering the original agent descriptions.

Any rules that have been derived by analysing the results of a single vignette, and which have been shown to be accurate within the domain of the vignette, need to be generalised into rules that apply across multiple vignettes. In other words, they need to be expressed in terms of how the behaviour of the SoS depends on the context provided by the vignette.

Rationale

It is not possible for an analyst or modeller to specify all important parameters ahead of time. Even if some hazard is influenced by an aspect of the model that *could have been* manipulated by an explicit parameter, it may be that the modeller does not implement that parameter. In that case, the nature of that hazard can only be discovered by studying the internal workings of the model.

Even if parameters that cause a given hazard are discovered and implemented ahead of time, it is quite possible for the accident-causing effect of one parameter to occur by multiple internal 'routes' through the SoS. For hazard analysis to be complete, all such routes must be identified.

If an analyst is to describe the mechanism by which a deviation leads to an accident, then they will need to understand the causation in terms of the behaviours of component systems, and possibly in terms of the internal workings of the those component systems. If an automated approach can provide at least part of this causal model up front, then it will greatly aid the analysis process.

Finally, the later stages of the safety lifecycle require hazards that are expressed in terms of the SoS, its component systems, and a concise description of its situation. Building the entire vignette description into each hazard description will make them far too unwieldy to be useful. In addition, hazards that are applicable in a particular concrete vignette may be exhibited in a wide variety of circumstances that the other vignettes have just failed to capture.

5.3 A Method for Finding Hazards using SoS Simulation Models

Given the requirements presented in Section 5.2, this section presents a method for analysing SoS models that have been developed using the method presented in Chapter 4. An overview of the process is given in Figure 5.2.

The steps of the method are:

1. **Acquire** a suitable simulation model for the system to be analysed
2. **Explore** the space created by the deviations that have been specified for the model
3. **Learn** hazard rules that relate combinations of deviations to accidents
4. **Explain** the identified hazard rules in terms of the behaviour of the agents in the simulation
5. **Validate** the hazards that have been identified and explained in the previous two steps
6. **Combine** the plausible hazards drawn from all the vignettes that have been analysed
7. **Feed back** the evaluation of the model's accuracy to the modelling process, so that it can be improved before the next analysis iteration
8. **Communicate** the results of the analysis to the rest of the SoS project team

These steps are explained in the sections that follow.

5.4 Step 1 — Acquire Model

The first step in an analysis process is, of course, to develop or otherwise acquire a model to analyse. The requirements for suitable models were given in Chapter 4, along with a process that, if followed, should lead to models that satisfy those requirements.

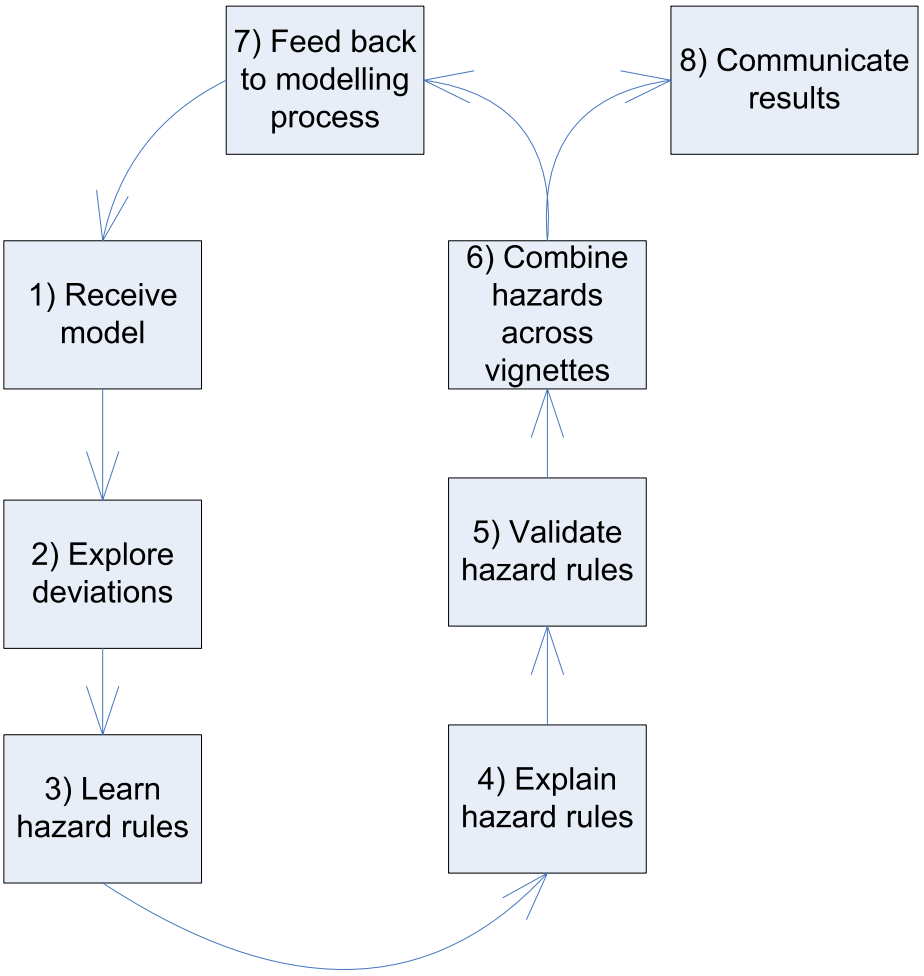

Figure 5.2: The Analysis Process

Potentially, a model *not* developed using the process described in this thesis could be subjected to the analysis presented here. If the model meets the identified requirements then this may be practical.

In either case, the quality of the model used is crucial for the validity of the overall analysis. The analyst must therefore assess the suitability of the model throughout the entire analysis process, and gather questions, comments and criticisms that can be provided to the modeller in step 7 (see Section 5.10).

5.4.1 AGO Case Study

The model for the AGO study was developed as a running example throughout Chapter 4. Although some criticisms of the implementation were noted in Section 4.13.4, it is sufficient for purposes of illustration. In step 7 (Section 5.10) the issues with the model will be discussed, along with any further problems that arise during analysis.

As noted in Section 3.3, for a given SoS it is relatively easy to determine the types of accidents that can occur, since the set of component system types is finite and there are only a few ways in which an component system can be involved in an accident. Simple examination of our model reveals that the following accidents are possible in the AGO system.

- Accident 1 — Helicopter collides with another helicopter
- Accident 2 — Helicopter collides with a UAV
- Accident 3 — Landed helicopter is hit by artillery fire
- Accident 4 — UAV collides with a UAV
- Accident 5 — Helicopter hit by enemy fire

5.5 Step 2 — Explore the Space

In order to meet requirements A1 and A2 while dealing with the problems identified in Section 5.1, a way must be found to explore the space of possible deviations while neither missing plausible hazards nor taking an impractically long time. The solution adopted in this thesis is to perform an exhaustive search through the powerset of combinations of deviations limited by a cap on the least probable combination that will be explored.

As noted in Section 4.9.4, the probability of a given combination of deviations can be calculated (with appropriate allowance of non-independence between deviations). The plausibility of this combination can be determined by setting a threshold for 'incredibility of failure'. This concept originally stems from the nuclear industry, and situations that appear to be more improbable than this threshold are not studied further in hazard analysis. A value for this is given in [130] as 10^{-7} per year of operation (equivalent to 10^{-11} per hour), and this value has been adopted for the work described in this thesis.

It can be noted that, given the early stage in the development lifecycle and the known use of low-fidelity models, the probabilities derived for deviation combinations are likely to be rough estimates. It may therefore be sensible to increase the threshold somewhat to allow for this inaccuracy.

Existing powerset algorithms can be used for this process, and algorithms exist that produce successive elements in the powerset in constant space and time.

An alternative to a powerset algorithm would be one that is specifically designed for this problem of generating all the alternative ways to 'spend' the 'probability budget'. This is not, however, the standard bin-packing problem (which involves finding the *single* optimum way to fit a set of items into *multiple* bins [131]), nor the knapsack problem (which involves finding the single highest value of items that can be fitted into a certain space [132]). It can be noted that any such algorithm that worked with simple additive or multiplicative deviation probabilities might not be valid for non-trivial probability combination rules.

Regardless of the precise algorithm used to achieve the goals given in this section, the selection of the next run to perform is independent of the results of previous runs, so although the (computationally cheap) generation of the powerset and evaluation of deviation probability needs to be performed in single process, the (relatively expensive) execution of the runs themselves is straightforward to parallelise and distribute.

5.5.1 AGO Case Study

For exploration of the AGO parameter space, a powerset of deviations was performed using an assumption of a flat 10^{-3} chance of each deviation occurring, and a minimum probability of 10^{-11} per run. In this simple case this means that runs with up to three deviations will be considered. To support the use of the machine learning techniques that were to be applied in the next step (see Section 5.6), runs of up to four deviations were performed in order to provide adequate data to learn from. With the five deviation types given in Section 4.9.9, and the set of agent instances that are present in the vignette, this results in a total of 4048 runs to be performed.

For the five accidents identified in Section 5.4.1, three of them manifest in this set of runs. Accidents 1 and 2, involving helicopters colliding with UAVs or other helicopters, do not occur in any of the runs we are working with. It may be that these accidents cannot realistically happen in the AGO SoS. However, it is also possible that there absence is due to the precise configuration of the model that was used (for example, the flight paths of the helicopters that resulted from the positions and plans of other agents). Section 5.6.1 will discuss the potential for using incident detectors to provide near-miss detection, thereby broadening the set of accidents that can be studied in a given simulation configuration.

5.6 Step 3 — Learn Hazard Rules

5.6.1 Detecting Accidents and Incidents

As noted in Section 4.5, identifying the specific *accidents* that can occur in a simulation model is relatively straightforward. These accidents should have been identified as concerns at the start of modelling, and have been incorporated in the implemented simulation as events that occur in the logging output. For the purposes of analysing simulation output, an ‘accident labelling function’ can be defined that takes a log event and identifies which, if any, of the possible accidents that the event corresponds to.

It can be noted, however, that accidents in the real world are rare events; accidents are frequently avoided by a distance of a few metres or by the unexpected occurrence of a chance event. It follows that in a simulation of reasonable fidelity accidents will also occur only rarely, and this could cause important hazards to remain unidentified.

A solution to this is to introduce ‘incident’ detectors that detect such “near misses”. Examples

include activations of an aircraft's collision-avoidance system, separation between two aircraft falling below some safe level, and queries to a superior of the form "Is X hostile?" when X is in fact friendly. Unlike actual accidents, a great many types of such incidents can be described. It can be noted that many incidents correspond to accident precursors; when an incident occurs, it may be the case that an accident could happen without any other deviation from normal behaviour. Alternatively, one can use incidents that are known to be correlated with the occurrence of accidents; for an example of such 'abstract safety indicators' in the automotive domain, see Svensson [133].

The major advantage of incidents over accidents is that incident detectors running in the simulation can be given variable sensitivity, so that they can be adapted to the level of risk that is actually exhibited in the system. For example, it is desirable to calculate actual collisions accurately, so that their effects on the unfolding scenario can be modelled realistically. Changing the sensitivity of an accident detector (such as by treating two aircraft as colliding if they came with two wingspans of each other) would allow accidents to be detected more easily but would have knock-on effects on the rest of that run. By comparison, an incident measure based on aircraft proximity can be as sensitive (or insensitive) as is required since triggering it only affects the statistics gathered, not the events that follow it.

One disadvantage of using incidents is that the simulation may have to be instrumented with purpose-built detectors, because it may not be possible to derive all desired incidents from the log file alone.

5.6.2 Discovering the Deviations that Lead to Accidents

As noted in requirement A2 (Section 5.2.2), once the parameter space of the system has been explored, the results must be analysed in order to find the relationships between deviations and accidents. It was observed in Section 5.1 that conventional statistical techniques were not particularly applicable to the hazard analysis problem. Alternate methods are therefore needed.

5.6.2.1 Defining 'Causes' Using Logic Programming

An approach that was initially explored by the author is to exploit the ability of logic programming systems to represent both the data (in this case, data about the events that occurred in a series of simulation runs) and the definitions about the relationships between concepts. By defining the term 'causes' in terms of the temporal relationships between events occurring within and across simulation runs, it should be possible for a logic engine to derive the causal relationships between events from a set of such runs.

If we define "X causes Y" to mean "X never occurs in a run without Y occurring later in that run", then a definition of the 'causes' in the Prolog logic programming language [134] is as follows:

```
causes(TargetEvent,CauseEvent) :-
       \+(happens_without_following(CauseEvent, TargetEvent)).

happens_without_following(CauseEvent, ConsequentEvent) :-
       event(CauseEvent,Time,Run),
       \+(happens_later_in_run(ConsequentEvent,Time,Run)).

happens_later_in_run(Event, Time, Run) :-
       event(Event,EventTime,Run),
       EventTime > Time.
```

Key: Prolog rules are in the form 'X :- Y', meaning 'X is true if Y'. Terms on the right-hand side joined by commas are ANDed together, and '\+' means 'not'.

It can be noted that, given the way that the Prolog language is typically implemented, such a definition is extremely inefficient. For practical application it was replaced by one that was more complex and less immediately intelligible. This is, however, of no concern here. Even with a reasonably efficient implementation, this did not scale well — it was practical for analysing run sets of a few hundred runs in size, but beyond that the time needed for processing increased rapidly.

It can be noted that the author originally attempted to use this method to learn causal relationships between accident events and prior events, rather than between accident events and deviations, so it is possible that it would scale effectively in this case. However, the approach is also unsuitable because of its sensitivity to 'noise' in the data; a single run where a potential cause is *not* followed by the consequent of concern will prevent that potential cause being identified, even if in all other cases the causal relationship holds.

5.6.2.2 Finding an Alternative

It followed from the attempted use of logic programming that something better was needed. Such an alternative had to cope well with non-linear, highly contingent behaviour (in effect, being able to find multiple 'routes' through the state space of the system). It had to be able to relate discrete-valued parameters to discrete-valued outputs, and it had to scale to large numbers of runs and find patterns that are less than 100% accurate.

The alternative that is presented in this thesis is the use of machine learning techniques.

5.6.3 Using Machine Learning to Identify Hazards

Mitchell, in [135], describes machine learning systems as *"computer programs that automatically improve with experience"*. Typically, a machine learning algorithm is initially given a 'training' data set consisting of input data items with their corresponding desired output. The

algorithm then produces a model that is much simpler than the training data itself, but that (ideally) captures all the information that is contained in the data. The model can then be interrogated — if the algorithm has learned well, the model will be able to provide the appropriate output when the corresponding input is supplied.

Machine learning has been applied to a wide range of applications; Mitchell [135] identifies successful examples as including speech recognition, control of an autonomous vehicle, classifying astronomical structures and playing backgammon. The most widespread use of machine learning techniques is to extract information from large datasets, such as those relating to a company's sales or the behaviour of financial markets. Such use is commonly termed 'data mining' [136].

Data mining can be seen as a way of finding patterns in data. This is what is needed in the current context: a way of finding patterns that relate deviations to accidents.

It can be noted that machine learning algorithms fulfil a role similar to traditional statistical techniques as discussed in Section 5.1.2. They have the advantage, however, that they provide a range of ways to express complex models of data that remain compact and comprehensible. Indeed, some algorithms can learn models that, for example, no regression technique could ever realistically derive.

5.6.3.1 Machine Learning as Function Approximation

An important way to think about machine learning is in the form of *function approximation*. In this view, a machine learning algorithm is learning a model that approximates the behaviour of some function. The algorithm is trained using a data set where each entry contains a combination of values for the function's parameters and the corresponding correct output from the function, and from this derives a model that behaves comparably to the function. In essence, the learner inductively generalises the function from the examples; this requires the use of a policy for generalising beyond the data (most likely implicit in the learning algorithm), known as the *inductive bias* [135].

In the terminology typically used for function approximation, the function to be approximated is the *target function* (which will have a result of a particular type, e.g. real-valued number), the parameters of the function are the *features*, and the combination of a set of feature values with the corresponding target function output is a *training instance*.

A typical example of function approximation would be learning a boolean-valued function for deciding whether an applicant for a mortgage or bank loan would default on their repayments. Each training instance could consist of a description of a person who previously took a loan with that institution. The features might include such personal details as age, occupation, income, credit rating, and the associated target function value could be a boolean value stating whether or not that person defaulted on the repayments. The learned function could then be used to judge whether a future loan applicant would be likely to default, and so inform the decisions of whether to accept their applicant.

Another similar example is used by Witten and Frank in [136], based on the data originally presented by Fisher in [137], where the features are size measurements made of different parts of an iris plant, and the target function returns the type of iris that the measured specimen is. Here the result is not boolean ("Is this an iris?") but rather a discrete-valued classification ("What type of iris is this?").

It can be observed that in neither of the above examples can the function that was approximated be said to 'exist' in the world prior to the learning — rather, the learner has provided a function that gives results comparable to such a function had it existed (at least within the subset of the function's input domain covered by the training set).

5.6.3.2 Function Approximation for Hazard Analysis

It can be observed that the simulation models developed using the process given in Chapter 4 take a number of parameters (boolean values representing the presence or absence of a given deviation) and produce as output (among other things) a list of the accident events that occurred during the corresponding simulation run. Such a simulation can therefore be viewed as a function mapping from the deviations to the resulting list of accidents. Given probabilities for the deviations, such a simulation could therefore theoretically be used to perform complete safety analysis for the modelled system by simply combining the results of all the runs, given the probability of each set of deviations occurring.

Unfortunately, the practicalities of real-world SoS modelling mean that the resulting safety statistics would not be credible. As discussed in Section 3.7 the practical role of the simulations discussed in this thesis is not to provide definitive predictions of SoS behaviour but to suggest hypotheses about how accidents can occur. The goal here is therefore to *comprehend* the "accident function", not to *use* it.

Although it can be said that the simulation model contains the information about such accident hypotheses, the complexity of the model is far too great for an analyst to comprehend directly. If this were not the case, the execution of the simulation model would not be necessary. A manual analysis of the model description — in the manner of, for example, a HAZOP (see Section 2.4.2.4) — would be sufficient.

We can, however, use function approximation to derive a simplified model that has (ideally) the same mapping of deviations to accidents as the simulation. Since such a function doesn't need to include all the internal complexity of the original model (e.g. the specific agent behaviours that combine to produce those accidents) it can have a much simpler structure and hence be open to manual comprehension. It can be observed that such a model is no longer open to modification (for example, modifying the behaviour of one of the agents) but that is not a concern as we can apply such changes to the simulation model and recreate the approximated accident function.

5.6.3.3 Available Machine Learning Algorithms

A wide variety of machine learning algorithms have been proposed; a survey can be found in Mitchell [135]. The algorithms differ in a wide range of ways including their internal knowledge representations, their applicability to different types of feature and function result, and the pattern of presentation of training instances (one by one or all at once) that they can support.

For the purposes described in Section 5.6.3.2, some criteria for the learning algorithm can be derived. The first requirement is that the algorithm be able to map from boolean-valued features to discrete outputs. Second, it must be possible to represent the learned model in a form that a human analyst can read and comprehend. Finally, because the algorithm needs to learn from a large number of simulation runs the algorithm must scale well to large training sets.

In the context describe here, the entire training set will be presented up front before the function needs to be examined. The ability to build the function progressively as data becomes available (which is often a concern, and is supported by some learners) is not required.

Reviewing the most prominent machine learning approaches, several candidates present themselves. Artificial Neural Networks (ANNs) [138] are a widely-used category of learner (indeed, they are perhaps the most widely reported method in the non-specialist media). ANNs are, however, unsuitable for the current purpose as the model they build is represented in a form that is generally unintelligible to humans — in effect, the model they build is a black box that can only be executed, not fruitfully examined. There have been some attempts to resolve this problem; see for example see Towell and Shavlik in [139] and Kurd in [140], however these are either limited in their expressive power or not implemented in generally available tools. ANNs are not, therefore, suitable for the current purpose.

Reinforcement learning (see Kaelbling et al. in [141] for an overview and survey) is another widely used technique, which is primarily concerned with developing the optimal policy by which an autonomous agent can take actions in its environment in order to achieve its goals. It can be seen that this setting has little in common with the hazard analysis task presented here.

Inductive Logic Programming (ILP) (described by Muggleton in [142]) is promising. ILP algorithms supplement the information gained from the training set with background information supplied by the user, potentially increasing their ability to make sense of complex data. They express their learned models using first-order logic, which is compact, expressive and human-readable. It can be noted, however, that the hazard analysis task as described in the previous section doesn't need the power of first-order logic in its rules; simple propositional logic is sufficient.

An ILP approach therefore sounds promising for the current purpose. However, published comparisons of performance between ILP and much simpler algorithms suggest that ILP performance can be very poor, and there is a lack of generally available tools that implement ILP in an easy-to-use fashion [143]. ILP is therefore not suitable for the work described in this thesis, although it may warrant further investigation in future work (see Section 8.2.5).

5.6.3.4 The C4.5 Decision Tree Learning Algorithm

Mitchell, in [135], notes that *"decision tree learning is one of the most widely used and practical methods for inductive inference"*. They can learn a wide variety of functions mapping from discrete features to a discrete output, and they build models (the decision trees themselves) that can be rendered in a human-readable propositional logic format. They typically require the complete training set to be provided before the tree is built, and cannot then be updated with further training instances, but as noted previously this is not a problem for the current purposes.

C4.5, presented by Quinlan in [144], is a widely-used example of a decision tree learning algorithm. Quinlan observes that C4.5 is suitable for function approximation tasks where the features are fixed in number, fixed in type (and constrained to simple discrete or numeric types), where the function result is of a discrete-valued type, where large volumes of training instances are available, and where the resulting model can be described by propositional logic. All of these properties hold for the current problem.

The concern can be raised that logical representations of knowledge (such as the propositional logic form used by C4.5) are inadequate for expressing complex patterns when compared to quantitative representations such as those used by ANNs. Fox and Das, in [129], question this concern, presenting an example from medical diagnosis where a logical representation was as effective as quantitative representations, and had the added advantage that it could explain its diagnosis in terms of the knowledge it had used. On this basis, it will be assumed that the decision tree representation is adequate for the current purposes.

The C4.5 algorithm (see [144] for the most complete exposition) builds the decision tree by successively partitioning the data, with each partition corresponding to a node in the tree. Each such node performs a boolean-valued test on one feature of the training set, and splits the training instances according to whether the result of that test applied to them is true or false. For example, in the current work (which uses boolean-valued features) a node might correspond to the test 'Deviation AILERON_JAM is true'. When creating such a partition, the algorithm chooses the test that provides the two most homogenous partitions. Such a test will provide the maximum information gain that is possible at that stage using the available features.

It can be observed that in the case where a feature in the training set contributes no information (i.e. that feature has no predictive value) then the feature will not appear in the learned tree at all. In this case, the algorithm has removed a redundant feature.

After the tree has been built, it is 'pruned' by removing subtrees, using a variety of techniques such as the 'reduced-error pruning' described by Quinlan in [145]. As well as increasing the accuracy of the tree, this makes it likely that parts of the model corresponding only to noise in the data will be discarded, and so will not distract analysts.

As an illustration of decision tree learning, consider the data in Table 5.1. This represents (wholly fictional) results that could have been generated by a real or simulated aircraft under several failure conditions. When given the data from the table, the C4.5 algorithm produces the

aileron_jam	autopilot_fail	reduced_visibility	Accident
no	no	no	no
yes	no	no	yes
yes	no	no	yes
yes	no	no	no
no	yes	no	no
no	no	yes	no
no	yes	yes	yes

Table 5.1: Example Data for a Decision-Tree Learner

tree shown in Figure 5.3.

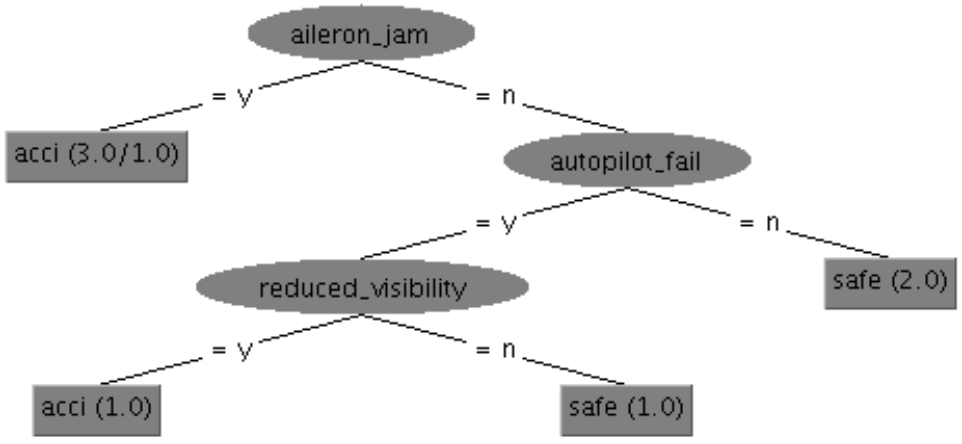

Figure 5.3: Decision Tree Learned From the Data in Table 5.1

This example shows how the decision tree learner can learn from probabilistic data (in Table 5.1, an aileron jam causes a crash in two cases but not in a third) and that patterns that involve two or more variables simultaneously taking on particular values can be learned (the loss of autopilot and reduction of visibility are only dangerous when combined).

In this example, all three features (the non-result columns) contribute information about the likely value of the result, and therefore appear in the learned tree. The complexity of the tree is therefore comparable to that of the table. In the more complex data sets produced in the case studies, however, many features were not of value and were excluded from the tree. In those cases, therefore, the algorithm provides a significant simplification of the data.

The decision tree presented in Figure 5.3 is simple and easy to understand. It has been observed, however, (e.g. by Michie in [146]) that large decision trees can be unwieldy and difficult to comprehend. Fortunately, as described by Quinlan in [144], a C4.5 decision tree can be converted into a set of production rules expressed using propositional logic. It was observed in Section 4.1.1 that a hazard can be described as a rule that describes how a combination of certain deviations with a given SoS vignette leads to an accident. Some rules that can be derived

from the tree in Figure 5.3 are:

- aileron_jam IMPLIES accident
- autopilot_fail AND reduced_visibility IMPLIES accident

Standard C4.5 is restricted to purely serial processing, which obviously limits its practical scalability — to build a tree from a given training set, only a single processor can be used. However, machine learning algorithms, including variants of C4.5, are an area of active development and research. For example, standard C4.5 as presented in [144] is suitable for only serial processing, but work towards parallelising C4.5 is presented in [147] and [148]. As noted in Section 5.5, the execution of runs can be parallelised, and parallel learning algorithms would allow analysis to be parallelised as well.

5.6.3.5 Issues in the Use of C4.5

A major decision when using C4.5 is selection of the training data to present to the learner. The most fundamental concern is that a large training set will lead to a complex tree, which in turn leads to complex rules (the mean length of the rules derived from a decision tree is the same as the mean length of the tree's branches.) Such complex rules are hard to understand and work with. A second concern (described by Chawla in [149]) is that C4.5 can easily fail to learn rules for minority cases (function output values that are only represented by a small proportion of the training set).

The solution adopted in this thesis is, rather than to learn a simplified model of the whole simulation, to learn one model per possible accident. In effect, the simulation model output is 'sliced' into multiple functions, one per possible accident, and then the learner used to approximate each of those functions. The training data subsets, and the resulting functions, each have a boolean output — true or false that the resulting accident occurred. Although all features are theoretically available to the learner, in practice it will only need a subset of them to provide an adequate approximation of the function, and will therefore produce much simpler rules.

One concern is that this slicing of the training set will lead to a very high ratio of safe runs to accident runs, and as noted above C4.5 may learn badly in this situation. One option is to drop many of the safe runs, but doing so arbitrarily may have unexpected results on the learned model. In practice, in the case studies performed we have not found this to be a significant problem.

5.6.3.6 Using the Learner

Once log files exist for simulation runs with all plausible combinations of deviations, as performed in step 2 (see Section 5.5), it is straightforward to apply the learning algorithm to this data. The deviations applied to each run are known (written to the log by the simulation engine) and the resulting accidents and incidents are either recorded directly in the log file (when

detected directly by the simulation engine) or derived by the accident labelling function (when only inferred from other simulation events). The analyst therefore has the data available to learn deviation-accident rules and creating an automated system that uses this data and learns all possible rules is straightforward (given an implementation of the learning algorithm).

This analysis system can easily produce a huge number of rules. The system must therefore filter and prioritise the rules so that it presents the analyst with (at least initially) those that are most important.

It can be noted, in this context, that the decision trees generated here are similar to the 'event trees' that are often used in manual hazard analysis (a description of these is given by Villemeur in [8], under the heading of 'The Consequence Tree Method'). When building event trees it is common to apply probabilities to the events/failures that occur and severities to the end events (accidents). A similar approach can be taken here, and the system can prioritise the rules based on the combination of their probability and severity. (Event trees are also relevant here in that the output of the learner can potentially be presented in that form in order for it to be more comprehensible to working safety engineers; see Section 5.11.1).

The probability of a rule is derived in exactly the same way as the probability of a run, as described in Section 5.5. It can be observed, however, that the rules produced by the learner may contain negative terms (for example 'NOT deviation1'). These could be taken into account in terms of probability of the rule by calculating the inverse probability of the positive rule (for example, the inverse of 1×10^{-3} is 9.99×10^{-1}). However, because these probabilities are so high, and because large numbers of negative terms will complicate the rules, these negative terms will be treated here as having a probability of one.

Severity is a property of an accident and therefore requires further attention. Severity can be classified in one of two ways. The first involves assigning an accident to a severity category based on a description of its consequences. This approach is used by several safety engineering standards and is probably the most common approach in the field. For example, MIL-STD-882D [11] defines 'mishap severity categories' of 'catastrophic', 'critical', 'marginal' or 'negligible'. Definitions of these categories are provided in Table 5.2.

Once severity has been assigned, it needs to be combined with a predicted frequency of occurrence in order to determine a risk classification. The risk classification then determines the acceptability of the hazard. Table 5.3 shows the frequency categories used in MIL-STD-882D. The standard does not provide specific values for the frequencies, requiring those to be determined on a per-project basis, and the values given in the table are a (plausible) example.

Table 5.4 shows how the severity and frequency are combined to provide the overall risk classification. Cells 1–5 (shaded in black) are 'high' risk, cells 6–9 are 'serious' risk, cells 10–17 are 'medium' risk and cells 18–20 are 'low' risk. The precise interpretation of these categories will depend on the context of the project; in the US military, the risk classification determines the level of authority required to 'sign off' (accept) the risk. For example, MIL-STD-882D suggests that a medium risk can be accepted by a Program Manager, but that to sign off a serious

Description	Category	Environmental, Safety and Health Result Criteria
CATASTROPHIC	I	Could result in death, permanent total disability, loss exceeding $1M, or irreversible severe environmental damage that violates law or regulation.
CRITICAL	II	Could result in permanent partial disability, injuries or occupational illness that may result in hospitalization of at least three personnel, loss exceeding $200K but less than $1M, or reversible environmental damage causing a violation of law or regulation.
MARGINAL	III	Could result in injury or occupational illness resulting in one or more lost work days(s), loss exceeding $10K but less than $200K, or mitigatible environmental damage without violation of law or regulation where restoration activities can be accomplished.
NEGLIGIBLE	IV	Could result in injury or illness not resulting in a lost work day, loss exceeding $2K but less than $10K, or minimal environmental damage not violating law or regulation.

Table 5.2: MIL-STD-882D Mishap Severity Categories, from [11]

Probability Ranking	Likelihood During System Operation	Probability (per hour)
FREQUENT	Likely to be continually experienced	$\geq 10^{-1}$
PROBABLE	Likely to occur often	$\geq 10^{-3}$ to $< 10^{-1}$
OCCASIONAL	Likely to occur several times	$\geq 10^{-5}$ to $< 10^{-3}$
REMOTE	Likely to occur some time	$\geq 10^{-7}$ to $< 10^{-5}$
IMPROBABLE	Unlikely, but may exceptionally occur	$\geq 10^{-9}$ to $< 10^{-7}$

Table 5.3: 'Frequency of Occurrence' Categories, consistent with MIL-STD-882D

risk requires the action of a Program Executive Officer.

An alternative approach to handling severity is to describe all severities quantitatively using a measure such as 'potential number of deaths'. This is clearly more demanding than using a small number of broad-brush risk categories, and raises a variety of issues such as the relative importance of injuries and deaths (although attempts have been made to express conversion ratios between deaths and injuries, and indeed one such attempt is embodied in version 'E' of MIL-STD-882 [150]).

The advantage of the quantitative approach for the current work comes when the analyst wants to combine probabilities and severities in order to rank the rules in probability order. For the risk category approach, MIL-STD-882D provides the risk matrix shown above in Table 5.4. However, Abrams and Cone note in [151] that the precise placing of boundaries on such risk matrix is contentious. It can also be noted (see Ekholm in [152]) that such tables do not allow risks to be combined across multiple hazards (in the current context, across multiple rules), and

		Hazard Severity Categories			
Frequency of Occurrence		I CATASTROPHIC	II CRITICAL	III MARGINAL	IV NEGLIGIBLE
A	FREQUENT	1	3	7	13
B	PROBABLE	2	5	9	16
C	OCCASIONAL	4	6	11	18
D	REMOTE	8	10	14	19
E	IMPROBABLE	12	15	17	20

Table 5.4: MIL-STD-882D Risk Matrix, from [11]

it is only by doing this combination that an assessment of the overall safety of the system can be made.

For the current work, there is a further problem in that there is no clear way to assign incidents to the accident categories, since they don't represent an accident or damage that has actually occurred.

The use of continuous severity measures provides a means to resolve these problems. Drawing on the financial-risk concept of Expected Monetary Value [151], the importance of a hazard can be measured in 'expected loss of life'.

For incidents, each occurrence can be considered to be a risk that an accident would have occurred in the real SoS. The severity of an incident can therefore be derived by taking the severity of the corresponding accident and multiplying it by a weight that corresponds to the probability that, in the real system, the incident would actually have been that accident. This weight will have to be assessed and assigned on an incident-by-incident basis. For example, an incident representing a near-collision between two aircraft in the simulation might have been judged to have a 0.5 probability of being an accident in the real system. If the combined number of deaths possible between the two aircraft was 60, then the incident would be treated as having a severity of 30 deaths.

5.6.3.7 Reconciling Effective Learning with Minimum Execution of Runs

It was noted in Section 4.9.6 that the modeller can reduce the space of deviation combinations that needs to be searched by ruling out certain runs ahead of time, for example by deciding that if deviation A is present then deviations B, C and D can be ignored. In a similar vein, as described in Section 4.9.5, allowances for the possibility of common cause may cause the engine to ignore runs that a systematic breadth-first search would have performed.

5.6.4 AGO Case Study

As noted in Section 5.4.1, it was possible to determine by simple examination the accidents that were possible. These then provided learning 'targets'; for each such target, a decision-tree model was learned to predict the combinations of failures that would cause that accident to occur. The list of the accidents that were used as learning targets, together with the labels used for them in the learning tool, was as shown in Table 5.5.

Accident	Label
Helicopter X hit by enemy fire	ehX
Helicopter X hit by friendly artillery fire	ghX
UAV X collides with UAV Y	cuXuY
Helicopter X collides with UAV Y	chXuY

Table 5.5: Possible Accidents

Enumerating these combinations for the different agents present in the AGO scenario gave 36 targets to learn models for. In practice, many of these accidents were not manifest in the available runs and therefore no rules were generated for them.

Table 5.6 summarises the results of this process. For each accident it shows the number of runs that exhibited that accident, a description of the highest-probability rule that was learned for that accident, and the probability of that rule (per run). The terms in the body of each rule correspond to instances of deviations from Table 4.14 being applied to specific agents. A key to the codes used for these deviations is given in Table 5.7.

Note that the table only contains those accidents for which examples were found in runs of probability 10^{-11} (per run) or greater. Many of the accidents that could potentially occur (as apparent from a simple examination of the simulation model) did not manifest in this result set. As discussed in Section 5.5.1, it may be that these accidents cannot realistically occur in the SoS, or it may be that the precise configuration of the model does not allow them to be revealed. For this case study, we will assume that these accidents cannot realistically occur. In a real-world application of the method, further investigation would be appropriate at this stage — the analyst would want evidence that the accidents were, in fact, impossible or wildly improbable.

It can be seen from the table that the most likely accidents are the loss of helicopters 2 through 4 to enemy fire, or collisions between UAVs 1 and 4 or 4 and 3. Most of the accidents that occurred have at least one rule that can cause them with a probability of 10^{-3} or better. Were this a real system, it is unlikely that this would be considered an acceptable level of safety.

As an example of a learned hazard rule, consider the case 'gh1' — 'an artillery unit destroys helicopter 1'. The decision tree learned for this target flattens into several rules, of which the three most probable are:

- lossofcommsfailure_uav4, !fusiongridnorthwestskew_uav2, !lossofcommsfailure_uav2,

Accident	Cases	Rule	Probability
eh1	2919	noairsensors_uav3	1×10^{-3}
eh2	3098	noairsensors_uav3	1×10^{-3}
eh3	2615	noairsensors_uav3	1×10^{-3}
eh4	3219	noairsensors_uav3	1×10^{-3}
gh1	36	lossofcommsfailure_uav4, !fusiongridnorthwestskew_uav2, !lossofcommsfailure_uav2, !fireskewnorthwest_gun1	1×10^{-3}
gh2	30	noairsensors_uav4	1×10^{-3}
gh3	27	lossofcommsfailure_uav4, !fusiongridnorthwestskew_uav2, !lossofcommsfailure_uav2, !fireskewnorthwest_gun1	1×10^{-3}
cu1u2	241	noairsensors_uav2, !fireskewnorthwest_gun1, !fireskewnorthwest_gun2, !fireskewnorthwest_gun3	1×10^{-3}
cu1u3	290	noairsensors_uav1	1×10^{-3}
cu1u4	877	noairsensors_uav4	1×10^{-3}
cu2u3	118	noairsensors_uav2, !fireskewnorthwest_gun3, !fireskewnorthwest_gun1, !fireskewnorthwest_gun2	1×10^{-3}
cu2u4	114	noairsensors_uav2, lossofcommsfailure_uav1	1×10^{-6}
cu3u1	290	noairsensors_uav1	1×10^{-3}
cu3u2	118	noairsensors_uav2, !fireskewnorthwest_gun3, !fireskewnorthwest_gun1, !fireskewnorthwest_gun2	1×10^{-3}
cu3u4	834	noairsensors_uav3	1×10^{-3}
cu1h1	25	noairsensors_uav1	1×10^{-3}
cu1h2	36	noairsensors_uav1	1×10^{-3}
cu1h4	25	noairsensors_uav1	1×10^{-3}
cu2h2	5	noairsensors_uav4	1×10^{-3}
cu2h3	97	noairsensors_uav2, fusiongridnorthwestskew_uav2	1×10^{-6}
cu2h4	2	noairsensors_uav4	1×10^{-3}
(other)	0	N/A	

Table 5.6: Summary of the Learned Rules

!fireskewnorthwest_gun1 → accident 1×10^{-3}

- lossofcommsfailure_uav4 !fusiongridnorthwestskew_uav2, !lossofcommsfailure_uav2, fireskewnorthwest_gun1, !fireskewnorthwest_gun2 → accident 1×10^{-6}

- lossofcommsfailure_uav4, !fusiongridnorthwestskew_uav2, !lossofcommsfailure_uav2, fireskewnorthwest_gun1, fireskewnorthwest_gun2, !fireskewnorthwest_gun3 → accident 1×10^{-9}

Code	Description
lossofcommsfailure	UAV loses ability to send out radio messages (hence, cannot update COP)
noairsensors	UAV loses ability to detect other entities in the air (hence cannot adjust course to avoid collisions)
fusiongridnorthwestskew	All entity updates performed by the UAV to the COP are 'moved' a fixed distance from their actual location in one of the cardinal compass directions
firespreadwidely	Effects of artillery fire are applied in a wider-than-usual area around the target location
fireskewnorthwest	The artillery perceives all entities in the COP as being 'moved' a fixed distance from their actual location in one of the cardinal compass directions

Table 5.7: Codes for AGO Deviations

5.7 Step 4 — Explain Hazard Rules

Meeting requirement A5 requires that, once the analyst has derived rules relating deviations to accidents, they need to explain them in terms of how the SoS mechanisms operate. This explanation is crucial for the necessary later steps of validating the detected hazards (see Section 5.8) and allowing changes of design or policy to deal with them (see Section 3.7).

A suitable explanation for a hazard needs to be at a lower level than the hazard rules from step 3, at (or near to) the level of the design model of the system as described in Section 4.8; it needs to be in terms of the internal mechanism of the model, rather than being purely in terms of deviations and the accidents that they cause.

An obvious approach to discovering this information is to derive a graph of all the possible paths through the vignette, starting with the baseline initial conditions and branching whenever the outcome of an event depended on the presence of a deviation. Such a graph could then be interrogated to find how events in the simulation model would lead to one another and eventually to an accident event. An example of such a graph is provided in Figure 5.4.

This is the approach taken by the Object-Based Event Scenario Tree (OBEST) method developed by Wyss et al. [153]. In this method, a simulation engine allows *"an analyst to probabilistically explore a 'universe' of possible scenarios that might arise from a set of initial conditions"* by executing a simulation model and branching as described above whenever a probabilistic node is reached. Wyss et al. note that this is similar to event tree analysis (see Section 5.11.1), but the use of object-oriented models and simulation allows much more of the system's behaviour to be captured.

There are problems with this approach, however. The resulting graph is potentially extremely large, and (unlike the event logs produced by exploring the space as described in Section 5.5) is monolithic. The graph will grow rapidly as the number of deviations, agents and recordable actions within or caused by the agents increases.

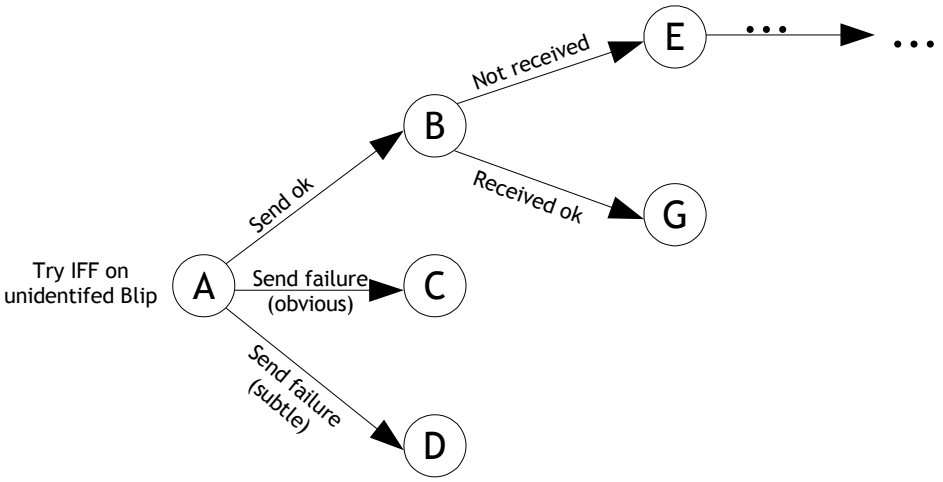

Figure 5.4: An Example of an Event Graph for a Simulation Model

Furthermore, in the event that circular causation is encountered, that part of the graph will grow indefinitely without ever terminating. Wyss et al. offer no solution to this problem in their published work.

It can also be observed that, in any case, the OBEST method is incompatible with the approach described in this thesis due to its assumptions about the nature of the agent models and the simulation engine that runs them. For example, it assumes that the modelled system is in continuous operation, and has a 'equilibrium' condition that can be disrupted by anomalous events. This is in contrast to the assumption in this thesis that each execution of the model is a vignette with a defined start and end.

5.7.1 Using a Tracing Tool to Find Accident Sequences

Lam and Barber, in [154] present a tool-supported approach to the comprehension of agent systems. The core of the approach is that, given a log of the events that occurred in a single simulation run, and an identified event of interest within that run, the system attempts to explain why that event happened in terms of its immediate causes. Each of those causes can then be explained in the same way, and the process repeated until the final explanation is in terms of the initial state of the simulation run or 'external' events that occurred. This explanation, complete or partial, can be expressed as a causal graph leading to the event that we asked the tool to explain.

A simple example of such an explanation would be of the form *"UAV 1 received a percept indicating the location of an enemy unit. This caused it to form a goal of destroying that enemy unit, which it selected the 'air strike' plan to resolve, and as a consequence of that plan the UAV conducted the 'attack' action using a laser-guided bomb"*.

The tool achieves this by storing what Lam and Barber call 'background knowledge'. This takes the form of a set of possible causal relationships between events of different types and different properties. As the tool tries to explain each event, it reviews these rules to find which previous events could have caused it.

For example, if the first rule in the background knowledge states *"A message can cause a belief if the data contained in the message appears in the data associated with the new belief, provided that the message arrives after the previous belief change"* then the tracing tool will try explain a 'new belief' or 'belief change' event as being caused by a prior message. If there are no messages, or no messages with suitable data, before the target belief and after the previous belief, then the tool will not find a causal connection using that rule and will move on to the next rule in the background knowledge.

Lam and Barber assume that the system to which the tracing tool will be applied was designed in terms of BDI (Belief-Desire-Intention — see Section 4.8) concepts. The BDI model defines the basic types of events that will occur during the running of the simulation, and hence provides a degree of structure for the resulting logs. A tracing tool can then exploit this structure.

It can be noted that the tracing approach is similar to OBEST in that it generates graphs of events and the causal relationships between them. However, rather than storing the whole graph for all possible (or performed) runs of a model, it generates a partial graph corresponding to an individual simulation run. This is possible because it exploits prior knowledge of the reasoning architecture employed by the agents in the system and about the causal relationships between individual types of simulation events. It is compatible with the rest of the approach proposed in this thesis because it can be performed retrospectively, using only the log file for a single simulation run.

5.7.1.1 Tracing for Hazards

Bosse, Lam and Barber, in [155], describe the use of a tracing tool to generate explanations of 'anomalous behaviour'. Such behaviour is identified because it violates assertions about the system that were laid down in the form of a temporal logic (TTL) expression. Once the unwanted or unexpected behaviour has occurred, the tracing tool is used to explain the events that comprise the violation.

It can be seen that this is directly analogous to the use of tracing tools to explain incidents and accidents. If the event to be explained is an accident, then the causal graph generated by the tracing tool is an accident sequence of the kind commonly employed in real-world accident analysis. (See Johnson in [36] for a presentation of accident analysis practices and techniques.)

The value of the tracing approach for the method described in this thesis is that it complements the machine learning described previously with detailed explanations of incident and accident causation at the single run level. Machine learning describes general rules that relate deviations to accidents; tracing shows how exactly the elements of the system and its environment interact to allow those deviations to cause those accidents. Once the learning algorithm has given

prioritised list of rules describing the ways that deviations can lead to accidents, the analyst can begin to work through the rules in priority order, taking a run that is covered by the rule and using the tracing tool to explain it.

Although the tracing tool concept is not restricted to event sequences expressed in terms of BDI concepts, the existing published work on the tracing method assumes this. This requirement on the output from the simulation model is quite natural if it is an implementation of a BDI design. If the model was not developed using a BDI architecture, it is still possible to use a BDI-based tracing tool. The model will have to be modified, however, to produce logs that express its behaviour in terms of BDI events. Effectively, this additional logging provides a BDI 'interpretation' of the model's behaviour. The ability to do this is valuable in that it may allow existing models to be adapted for agent tracing.

A set of agent concepts used for one application of the tracing method (to a simulation of a swarm of unmanned air vehicles performing a surveillance mission), taken from [12], is shown in Table 5.8.

Concept	**Instances**
Message	Messages about preferences, commitments and scanned targets
Belief	Belief about an agent's own preferences, commitments, and scanned targets and, via communicated messages, beliefs about other agents' preferences, commitments, and scanned targets
Intention	Ordered list of committed targets
Action	Fly to target, spiral (to search for a target), and stop
Event	A target is scanned (if the agent is within range as determined by the simulation)

Table 5.8: Example of Concepts for a Tracing Tool, from [12]

The tracing approach can be explained by an example. Taking the simple example of a log given in Table 5.9, the tracing tool can be used to form an explanation of the last event ('friendly helicopter destroyed'). ISTAR, UAV and HELI are all blue force (friendly) entities.

Some general-purpose rules can be assumed:

- Any action can be explained by the most recent plan adopted by the agent at the time of the action
- A belief is caused by a preceding message if the message occurs after the previous belief, the data of the belief contains the data of the message, and the recipient of the message and the adopter of the belief are the same agent
- The event 'agent destroyed' can be explained by a previous attack message naming that agent

Time	Agent	Type	Data
0	UAV	Goal	Eliminate enemy
0	UAV	Belief	No friendly aircraft in region
31	HELI	Percept	Blue force truck TRUCK1 at (x,y)
36	HELI	Belief	Reached current waypoint
37	HELI	Belief	No more waypoints
59	ISTAR	Message Out	Red force helicopter HELI at (x,y)
60	UAV	Message In	Red force helicopter HELI at (x,y)
60	UAV	Belief	Valid target (Red force helicopter at (x,y))
61	UAV	Plan	Attack target
62	UAV	Plan	Move into range of target
64	Command	Message Out	Assign FIGHTER1 to CAP
65	FIGHTER1	Message In	Assign FIGHTER1 to CAP
68	UAV	Belief	In range of target
69	UAV	Action	Fire missile at agent HELI
71	HELI	Event	Destroyed by UAV

Table 5.9: Example Log for Tracing Tool

The simple rules given above will explain that the belief of a valid target was caused by the message, and that the helicopter destruction event was caused by the 'fire missile' action, which in turn was caused by the 'attack' plan. It cannot, however, connect those two together. To help with this, the specific agent under consideration (the UAV) can be examined and some rules created that explain its individual behaviour. Imagine that inspection of the UAV agent's behaviour led to the following two rules that could explain the adoption of a plan to attack a target:

- An 'attack target' plan can be caused by the same agent having a goal to eliminate enemy AND the belief that a target has been positively visually identified
- An 'attack target' plan can be caused by the same agent having a goal to eliminate enemy AND a belief that a target is hostile AND a belief that there are no friendly entities of the correct type in the area

With these additional rules, the destruction of the helicopter can now be explained with the graph shown in Figure 5.5. A key to the graph (and to the other tracing tool graphs presented in this thesis) is given in Figure 5.6.

Notice how, although we have had to create a rule that is specific to a single entity, this rule is applicable in many possible circumstances. The rule is therefore a reasonable allocation of effort, because it has the potential to (partly) explain the behaviour of the UAV in many situations. Notice also how the explanation is generated using the minimum set of events necessary, and that the irrelevant events in the log have been ignored. For large logs this is extremely valuable.

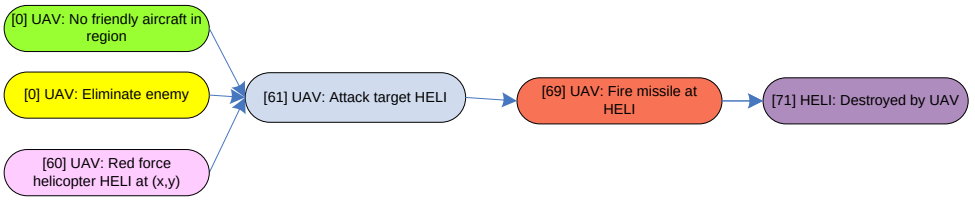

Figure 5.5: An Example Tracing Tool Explanation

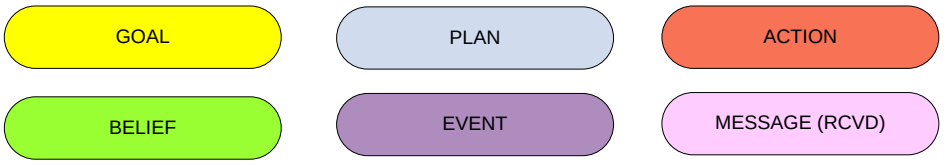

Figure 5.6: A Key to the Tracing Tool Graphs

Now that the immediate causal sequence has been explained, it would probably be possible to add additional rules to explain why, for example, the ISTAR decided that HELI was hostile in the first place.

A major strength of the tracing tool, as noted by Lam in [154], is that the resulting causal graphs are expressed entirely in terms of BDI concepts. This has two principal advantages when used with the modelling approach described in this thesis. Firstly, since the design model is expressed in terms of a BDI architecture, the graphs can be directly related back to the beliefs, plans and other BDI elements identified in the design model, regardless of the form taken by the operation model and the actual implementation. Secondly, as noted in Section 4.8, BDI descriptions of agent behaviour are readily comprehensible to non-specialists, including the domain experts who will ultimately have to evaluate the plausibility of the explanations provided (see Section 5.8).

5.7.1.2 Developing Rules for a Tracing Tool

The development of rules for use by a tracing tool are often quite simple, and follow naturally from knowledge of the system. In theory, particular attention does not have to be given to relating individual rules to one another; each rule only needs to take account of knowledge about a single behaviour or property of a single agent. It can be noted that this is similar to the way that multi-agent simulation allowed the event sequence to be generated in the first place using only descriptions of the individual agents, as discussed in Section 3.6.

Furthermore, development of tracing rules can be iterative. If an event in a sequence that seems interesting is not explained by the existing rules, the analyst can go back to the individual agent description and derive a new rules capturing conditions that could cause the agent to exhibit that behaviour. This process can begin with a single accident event, with the analyst seeking

new rules to progressively expand the sequence further back through the log. If these rules are sufficiently general (in that they apply to multiple situations or to multiple types of agent) then they will have value when tracing other runs or other agents in the same run.

It could be suggested that, since the modeller will have developed the behaviours of the agents that allow their actions to be determined (in effect, the rules that allow them to be moved forward in time), it should be possible for them to also provide the 'reverse' rules that allow agent behaviour to be explained. One might then suggest that a single source representation should be used that then generates both forwards and backwards rules.

Treating the 'reverse' rules used by a tracing tool as an additional product to be developed later, however, has several advantages. First, it allows the use of black-box agents (i.e. agent models provided without any source code) and non-BDI agents, provided that they output suitable logs. Second, this opens the door to a wide range of sources of input, including, potentially, actions driven by human input (although this is well outside the scope of this thesis).

If the reverse rules cannot be generated automatically (for example, because agent behaviour is described in imperative form, as is the case in many agent frameworks such as MASON [121] and Jadex [116]), then considerable programming effort may be required to create rules. In this case it is natural that rules only be derived as they become needed.

Regardless of the method by which the rules are derived, the selection of rules to use in any specific instance of tracing tool use requires some thought. Using all rules that are applicable to the system in question may substantially increase processing time, especially for larger logs, without contributing to useful explanatory power. Trying additional rules to explain each event takes time, and some individual rules will be computationally expensive (particularly those that search for multiple causes).

Rules may be either generic or specific to a particular domain. Lam, in [12] gives several examples of domain-specific rules:

- *If a 'uavScan' event e occurs after a 'flyToTarget' action a for the same target, then a caused e.*
- *If a 'scannedTarget' belief b occurs after a 'uavScan' event e for the same target, then e caused b.*
- *If a 'flyToTarget' action a occurs after a 'scannedTarget' belief b for different targets, then a is caused by b (i.e. the agent believes it has scanned its previous target and is pursuing its next target as prescribed by its intention).*

5.7.1.3 Tracing Tool Requirements on Log Files

The quality of the analysis provided by a tracing tool depends heavily on the quality of the rules used (as discussed in the previous section). Equally important, however, is the quality of the logs that it is applied to.

As noted earlier, the tracing tool discussed in this thesis relies on logs expressed in BDI terms. Log events that are not consistent with that architecture will have to be excluded from the causal graphs. This may mean that some exhibited behaviours cannot be explained.

It is likely that, in practice, iterative development of logging code will be needed; during an attempt to trace a particular agent behaviour, it may become apparent that a particular important event is not being logged. It will therefore be necessary to augment the simulation implementation with additional logging code and to repeat the runs that are of concern. As the tracing tool only works with one run at a time (rather than with large parts of the run set, as the decision tree learner does) it should be necessary to repeat only a small number of runs.

The solution to logging problems is not always to increase the amount of logging. Large volumes of log output will reduce the performance of the tracing tool, and will make it harder for the analyst to study the logs manually while developing new tracing rules.

The causal graphs that are output by the tracing tool will be used directly by engineers, so they need to be comprehensible to humans. If the granularity of the log events output by the model is too fine "e.g. ('turn left 1 degree', 'turn left 1 degree', 'turn...')", the tracer will tend to generate explanations that are too large and complex to be intelligible to an analyst. (It can be noted, however, that after-the-fact post-processing may be able to help with this kind of problem, for example by 'smoothing' long chains of movement actions into a single summary event (such as 'turned left 32 degrees' or 'turned to face the next waypoint').

5.7.2 AGO Case Study

Section 5.6.4 presented a set of rules for accident 'gh1' — 'artillery hits helicopter 1'. An initial manual inspection of the rules reveals that rule 1 is an artefact — the single deviation on UAV4 is not sufficient to cause the accident. The learner derived this rule because that deviation on UAV4 is otherwise highly correlated with the occurrence of the accident; based on the training set, the presence of the deviation is good reason to expect the accident to occur.

Rule 2 provides an explanation — in order for the loss of communications by UAV4 to cause a 'gh1' accident, one of the artillery pieces (in this case, gun1) must have its fired skewed to the northwest.

It can be noted that rule 3 is valid but is subsumed by rule 2, so will not be examined further here.

So, it has been observed that rule 1 is an artefact, and that rule 3 is subsumed by rule 2, but that rule 2 is valid and needs to be investigated. The tracing tool can now be applied to help explain these, provided that some rules specific to the AGO situation are supplied. Some examples are:

1. A message about an enemy causes a belief about that enemy when the location and the enemy strength of the message match those of the belief

2. A helicopter movement action, or an artillery fire action, causes a 'helicopter destroyed'

event if the coordinates of the cause match the coordinates of the event

A tracer explanation of the accident is shown in Figure 5.7.

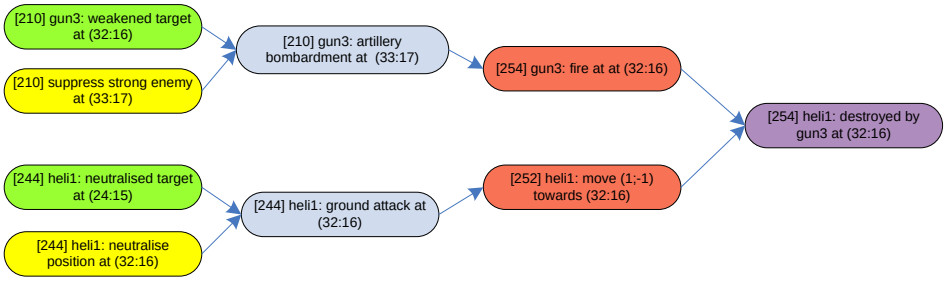

Figure 5.7: Tracer Results for Accident 'gh1'

It can be seen that the helicopter adopts the goal of neutralising anything at a particular coordinate and so moves to it. The gun, at the same time, adopts the goal of suppressing the enemy in the neighbouring square and fulfils the goal with plan of artillery bombardment. But the action actually performed (due to the 'skew' deviation applied to the gun) is to fire at the square that the helicopter has just moved into. The artillery fire therefore hits the helicopter.

Observing some of the runs which match rule 2 and where the 'gh1' accident does occur, it is apparent that removing UAV 4 from the data fusion loop (through communications loss) has an effect on the order in which enemy positions are detected, and that this affects both the order in which the guns target the enemy positions and the order in which the helicopters fly out to them. If the artillery are firing accurately and in the correct coordinate system, then this is still perfectly safe. However, in the case where an artillery piece has its coordinate system skewed in just the wrong direction, the change of movement order means that a helicopter can be hit.

This example is interesting in that it highlights an extremely subtle interaction ('coupling' in Perrow's terms) between component systems that could not easily be predicted. Once predicted, such an interaction demands investigation of the real system behaviour.

5.8 Step 5 — Validate Identified Hazards

The combination of the machine learning and the tracing will have lead the analyst to derive a number of rules that relate deviations to accidents and that provide tentative causal explanations for why they hold. The process has gone from the initial domain model (see Section 5.4) and its associated concerns (see Section 4.5) to a new set of concerns that are supported by a detailed simulation model. The category of simulation model with which this thesis is concerned, however, is still a long way from the real system. The new concerns must therefore be validated using a variety of means.

5.8.1 Use of More Detailed Simulations

One way to progress further is by the use of more detailed simulations. Although the original model will have been developed to have an adequate representation of the concerns originally identified, the new concerns will lead to different requirements. Additionally, the wide scope of modelling required by the approach described in this thesis will inevitably require the use of relatively low-fidelity models. A model built to evaluate a specific concern will be able to achieve higher levels of detail and accuracy.

For example, if one of the hazards identified was a collision between two UAVs, the analyst might validate this by modelling some of the known collision situations in a general-purpose synthetic environment, using a more detailed model of the UAVs. This model might include high-fidelity physical dynamics (3- or 6-degree-of-freedom movement, and atmospheric effects) and a better representation of the agent's behavior (an early prototype of the UAV's actual flight-control logic running in the loop). The analyst would not need to re-implement the whole vignette and all the other agents, even if those other agents were known to be instrumental in causing the collisions to occur. Rather, the analyst would just need to set up *the stimuli on the UAVs* equivalent to that which caused the UAVs to collide in the original model.

If the accident can be duplicated (or at least nearly duplicated) in the high-fidelity model then it is worthy of further consideration.

5.8.2 Review by Domain Experts

It is also possible to submit a newly-revealed hazard to domain experts for review. Done casually or naïvely, however, this can be misleading; such experts may casually reject a hazard sequence that seems intuitively implausible (given their theoretical perspective and previous experience). This is of course unacceptable, given the belief established in Section 1.2 that hazard analysis in SoS is not straightforward and is not generally tractable by manual analysis. It is central to the motivation of the work described in this thesis that the manual effort of engineers is not sufficient to identify and validate all risks.

The role of domain experts here is, therefore, to provide *reasoned arguments* for why an identified hazard is implausible. Such an objection is valuable because it can be validated; the analyst can explore (by simulation or by other means) whether the stated objection would remove the hazard (or make it statistically implausible).

For example, in following on from the UAV collision example described above, a domain expert might object that the original simulation did not simulate the supervisory role of the UAV ground controller, who would have detected the impending collision and given orders to prevent it. To address this objection, the analyst could augment the more sophisticated model with a simulation of the ground controller's behaviour, or could perform a study using another means to evaluate the likelihood of detection by that controller in the situation that the hazard describes.

Again, the key is that the evaluation of the hazard takes place using models (whether simulation or on paper) that are more accurate than the original hazard analysis model but that cover a smaller space and time frame and use only parts of the behaviour of a subset of the agents. The hazard analysis process described in this thesis makes this possible by 'sweeping out' the trajectories possible in the system and drawing attention to those that appear dangerous enough to warrant further investigation.

5.8.3 A Caution on Interpretation of Multi-Agent Simulation Results

It is important, in interpretation of the results of hazard analysis simulation, to exercise caution. Richardson, in [126], discusses the general problem of modelling nonlinear systems: it is very difficult to have confidence in extrapolation beyond the observed data. Without a high degree of confidence that the behaviour of a property of interest follows some kind of simple underlying model throughout its range, the validity of a result from the model that has not been also observed in the real system is highly questionable.

It can also be observed that multi-agent simulation is a relatively immature technology, particularly when compared to alternative simulation approaches (such as system dynamics approaches, the optimisation techniques used widely in operations research, and the equation-based models that have a centuries-long history in the natural sciences). Relatively little is understood about how validate multi-agent models, or how to assess their reliability outside the range in which they were originally validated. Edmonds notes in [127] that published academic work on multi-agent modelling is dominated by suggestions for modelling approaches and formalisms, with relatively little work on validation with respect to the actual systems that are being modelled.

In a similar vein, Humphreys (in [156]) observes that multi-agent simulations have a high 'epistemic opacity' in that it is often difficult to see how the descriptions of the individual agents and the environment (about which the modeller is perhaps quite confident) give rise to the behaviours observed at the system level. One particular risk he notes is that because it is often possible to tune the parameters of a multi-agent model until it produces behaviour consistent with the modelled system, the analyst may feel that they understand the system when they have identified a set of conditions that are *sufficient* to give the observed behaviour. This kind of explanation is inadequate; the analyst needs to understand the conditions (and, indeed, the whole range of conditions) that are *necessary* to cause the behaviour of interest.

These points can be illustrated with reference to hazard analysis simulation. It is not sufficient to produce a model that *can* exhibit a particular accident, nor to identify a set of combinations deviations that *can* cause that model to exhibit that accident. Rather, the analyst needs, eventually, to have at least a partial, mental model that is consistent with the real system (at least as far as that particular accident is concerned) and to know all the ways that combinations of deviations and conditions can lead to that accident occurring. It can be observed that the latter standard is a very demanding one.

Much of the controversy noted above, however, stems from the social sciences where multi-agent simulations are being (enthusiastically) pursued in search of scientific results with very broad applicability. Safety engineers have a task that is rather easier than the social scientists. First, engineered entities (such as radar stations, data fusion systems, and UAVs) are much simpler than the highly complex organisms (humans) with which social scientists are primarily concerned. Similarly, managed organisations (such as the human element of SoS, and military units) are under more complete control than open ones (such as the towns and countries that often concern social scientists). Finally, the scope and ambition differs: the social scientists want 'scientific' results that are broadly applicable with respect to different situations and different times; in the current work we are concerned primarily with engineering results that apply to one particular SoS for the duration of its operational life. It can be noted, however, that learning general lessons is very valuable for derivation of organisation-wide safety policy.

In general, when modelling and understanding complex systems and systems of systems there is a need to incorporate the perspectives of many models and many experts. Richardson [157] refers to this as 'pluralism'. The process and models described in this thesis cannot provide sufficient information for complete identification and evaluation of hazards. It can, however, provide stimulation for considering situations and phenomena that humans have difficulty comprehending with conventional manual tools. (See Resnick [38], for some examples of the difficulty humans experience in comprehending decentralised phenomena.)

5.8.4 General Questions of Fidelity

A standard objection to the use of simulation for analysis is to question the fidelity of the model with respect to the real system that it purports to represent. In this context, it is important to note that almost all hazard analysis is performed with respect to some model of the real system; it cannot be said to be performed on the system itself. This is partly because the complexity of a real system is unmanageable (and much of it irrelevant) but also because hazard analysis is important very early in the safety lifecycle, before the detail of the final system design is available.

Whilst there will always be concerns with the fidelity of the models we use, the use of models and approximations remains an inevitable part of real-world hazard analysis. One difference when using models for simulation, rather than for manual analysis, is that in manual analysis there is great opportunity for pragmatic human interpretation, thereby covering a multitude of deficiencies in any modelling approach adopted. In effect, the approach described in this thesis moves this interpretation from the middle of the process to the end — engineers are only able to effectively review the model once it has been executed and produced some results.

The use of simulation in this approach is for what Dewar et al. describe in [56] as 'weak prediction'. They note that *"subjective judgement is unavoidable in assessing credibility"* and that when such a simulation produces an unexpected result *"it has created an interesting hypothesis that can (and must) be tested by other means"*. In other words, when a simulation reveals a

plausible system hazard, other, more conventional analyses must be carried out to determine whether it is credible in the real system. The role, therefore, of the simulation analysis is to narrow down a huge analysis space into one that is manually tractable.

5.8.5 AGO Case Study

Given the explanation presented in Section 5.7.2, is it plausible that this could occur in the real world, or are we merely seeing a simulation artefact? It can be observed that the presence of this accident hinges on some fairly precise space and time coordination. This is a common phenomena in the types of accident considered in this thesis — an accident is often (in a superficial examination) only a hair's breadth from a near miss.

It can be observed, however, that the fire from the UGVs and the movement of the helicopters are concentrated around specific locations (those occupied by the enemy positions) and specific times (when the enemy are first revealed as valid targets). The kind of space and time synchronisation required for this accident is part of the normal operation of the system. Indeed, this synchronisation is *desirable*, in order to minimise the time between cessation of artillery fire and arrival of ground forces.

It can also be observed that there is no mechanism in the system as described for detecting or correcting problems of inaccurate artillery fire. The MODAF model shows fusing of target data and the results of infantry action on the ground, but nothing about feedback on the accuracy of artillery fire. This accident is therefore of concern and needs to be investigated further.

5.9 Step 6 — Generalise the Results Across Vignettes

The analysis method thus far has derived rules expressed in terms of deviations applied to a single vignette. In order to be general hazards applicable to the system, the rules need to be converted into a form that incorporates the aspects of the vignette that are involved in the causal chain of the hazard. In effect, the generalised form of the hazard is the vignette-specific rule combined with part of the vignette description.

For example, take a vignette where multiple UAVs are patrolling an area and are controlled by a single ground station. It might be determined that by combining the vignette deviation "poor visibility due to fog" with the agent deviation "reduced processing power" on the ground station might allow two UAVs to collide. If this was determined to be a realistically plausible hazard within the scope of the single vignette, it would then be converted into a general (vignette-independent) form. It might be that the collision could be shown to only ever occur for two UAVs that had an overlapping patrol area, which is of course a feature of the specific vignette. The final hazard can then be expressed as:

(UAV X and UAV Y have overlapping patrol areas AND visibility is poor AND ground station has reduced processing power) → (UAVs X & Y collide)

5.9.1 AGO Case Study

The accident described in Section 5.7.2 is possible because, as well as UAV4 having lost communications and gun1 having its fire skewed, the distribution of the enemy targets within the space is such that two enemies are adjacent, and it is therefore possible for a helicopter landed at one location to be hit by fire aimed at the other. Significantly, the effect of the UAV4 deviation is not direct — it merely changes the order in which enemy targets are attacked. Since nothing in the system description suggests that any such ordering is preferred, the presence of the UAV4 deviation can be considered a normal case rather than a deviation case.

The vignette-independent rule is therefore:

- fireskewnorthwest_gun1 $\wedge$ (AGO SoS tasked to attack targets of which two or more are in close proximity) $\rightarrow$ accident

Since the rule is composed of a single deviation on a single component system and an environment configuration that is perfectly plausible, it is clearly not acceptable, and must therefore be taken forward for further analysis.

5.10 Step 7 — Feedback to Modelling Process

As noted in step 1, as the analysis process continues an analyst should be aware of possible problems with the model that may need to be resolved. It is not sufficient for the analyst to "trust the modeller" and treat the model as authoritative — they must bring their experience and critical intelligence to bear on its evaluation. Sargent, in [77], observes that no simulation model of a non-trivial system is going to be entirely accurate on the first attempt, so analyst feedback is crucial if the next model iteration is to resolve problems in the current one.

If the model is at odds with the analyst's understanding of the world and engineering judgment, they may propose a change in the model, or a change of scale or abstraction (in order to avoid repeatedly analyzing equivalent phenomena, or to drill down into detail when some prediction cannot be made at the current level of the model).

As discussed in Sections 4.9.4 and 5.5, there is a need to identify effective heuristics for selection of simulation runs. As part of the feedback process, the heuristics being used should be evaluated and potentially changed.

5.10.1 AGO Case Study

The analysis of the model presented in this chapter has not raised any particular issues or problems with the model, but it has identified two particular details on which an accident appears to hinge. These need to be further evaluated.

First, it was noted in Section 5.8.5 that there is no mechanism or process specified by which the artillery could become aware of inaccurate fire and take steps to correct it. The modellers need to consult with domain experts and other sources to determine if this is realistic given the current SoS concept.

A second area for investigation is the precise targeting criteria. Section 5.8.5 notes that it is the time and space simultaneity of helicopter and artillery action that makes the identified accident plausible. However, Section 4.6.3 observed that the targeting criteria that lead to this were not specified in the source model and therefore had to be assumed. Since these appear to be critical for the occurrence of accident, the criteria need to be reviewed by the modeller prior to taking the results of the simulation further.

Another concern is whether the helicopters would and could take account of enemy position data when plotting their flight paths, thereby potentially avoiding exposure to enemy fire. Their possible use of low-confidence enemy reports (below the threshold for triggering an artillery barrage) is a particular concern since it is not obvious to what degree they would take account of these.

Finally, it can be noted that the accident "UAV collides with helicopter" did not manifest in any of the simulation runs. This is a valid result — it may be that the SoS as described does not make it plausible for this accident to occur. Given that this accident was identified as an explicit concern (see table 4.5), it is important to review the model to check if it has been unreasonably ruled out.

5.11 Step 8 — Communicate Results

In order for the results of the analysis to be useful, they must be communicated effectively to the rest of the SoS project team, so that they can act on this new knowledge. Failure to preserve, or communicate in the first place, the results of the hazard analysis would mean that the effort expended has been wasted.

Manual hazard analysis techniques often communicate their results by a document known as a *hazard log* that lists the identified hazards, their causes, their likely effects and other details such as an estimated probability of occurrence. Given the complexity of the hazards that are likely to exhibited by SoS, however, such a format is unlikely to be satisfactory. It will therefore be necessary to use a variety of formats for the description of SoS hazards.

A crucial part of the audience for hazard information is the group of other safety engineers who are working on the SoS, either as part of the SoS owner or as part of one of the component system owners. They will largely be responsible for determining responses to the hazards identified by the simulation analysis, and for disseminating the results of the analysis to non-safety-specialists. It is therefore desirable to present the results of the machine learning and tracing analysis described in this chapter in a form that they are familiar with.

To this end, it can be noted that the decision trees described in Section 5.6 are similar in form

to the common safety artefacts Event Trees and Fault Trees.

5.11.1 Event Trees

Event Tree Analysis is inductive hazard analysis technique, which creates a tree with an *initiating event* at the root and consequences of that event at the leaves. It is drawn with the root on the left edge of the page, with columns across the page representing possible subsequent events. At each column, each branch of the tree may branch again, showing the effect of this event occurring or not occurring. Each branch eventual leads to a possible consequence.

Villemeur, in [8], provides an example Event Tree for a broken pipe in a nuclear power plant. This is shown in Figure 5.8.

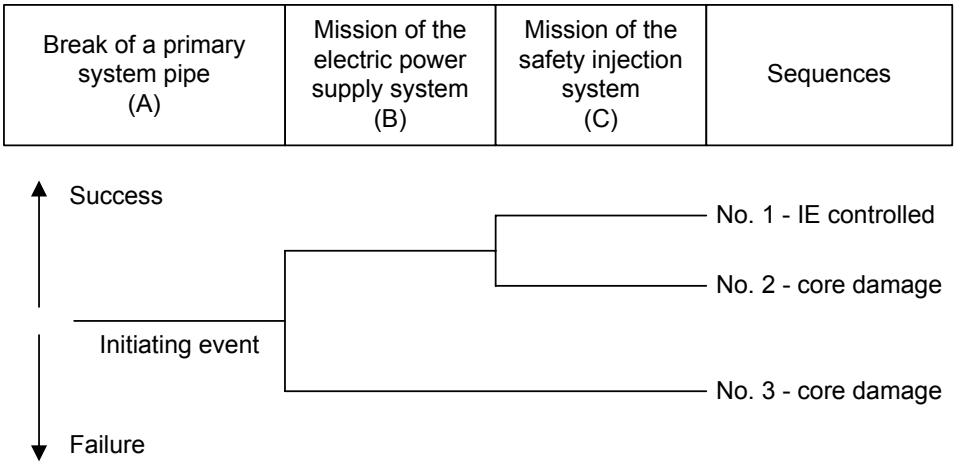

Figure 5.8: An Example of an Event Tree (from [8])

It can be noted that an event tree serves the same purpose as the hazard analysis simulations discussed here, in that it allows the accident behaviour of the system under certain deviations to be derived. However, an event tree represents a purely linear sequence of events, whereas the simulation models can capture complex non-linear behaviours and relationships, and they encapsulate sufficient detail that changes can be made to the system and the new consequences derived. In this, the simulation model wholly subsumes the event tree. It can be noted that this was part of the motivation for the OBEST method discussed in Section 5.7 — Wyss et al. observed that event tree analysis was valuable, but the standard notation and manual method severely limited the system behaviour it could express [153].

For the purposes of comprehension, however, the event tree is much more tractable, and has the added advantage that is widely understood among safety engineers. It is therefore desirable to derive event trees from the simulation.

The decision trees (and their derived rules) that result from the analysis described in Section 5.6 contain part of the information needed to generate event trees. Both event trees and hazard rules describe how a set of deviations from expected behaviour (in the case of conventional

event trees, only explicit *failure* behaviour) lead to accident that can occur. Event trees use the explicit concept of an initiating event for the start of the tree; for current purposes, this event can be the start of the identified vignette.

The hazard rules that are extracted from SoS simulations, however, are only partial — they show how *some* combinations of deviations lead to *some* accident results. The rules corresponding to multiple accidents need to be combined, so that all possible accidents are represented in the tree. Combinations of deviations for which the hazard rules suggest no accident will occur are not a problem. This assumption, of course, needs to be evaluated once the tree has been created, but the resulting tree provides a starting point for consulting with other safety engineers.

A more serious problem relates to the assumption of temporal ordering in the conventional event tree. Moving across the columns from left to right is moving forwards in time; in Figure 5.8, the pipe breaks before the condition of the electrical power supply is of consequence. In [8], Villemeur shows how this can be exploited to take account of proof test frequency and speed of repair after a problem is detected. If this is *not* exploited, it is possible to draw an event tree without temporal ordering, but such a tree may confuse safety engineers who assume such ordering.

Hill, in [158], notes that event trees developed for complex systems can easily become difficult to maintain and understand. For the current purpose, this complexity can be managed by producing event trees that are partial in their coverage of deviations or accidents. It is crucial, of course, that their partial nature be clearly indicated to avoid assumptions of completeness.

Overall, it can be seen that while the simulation fulfils much the same role as an (intractably complex) event tree, analysts must exercise caution when deriving an event tree from the simulation results.

5.11.2 Fault Trees

A fault tree is a diagram that shows how a set of possible events (typical representing faults occurring in some system) can lead to another event that has undesirable consequences (typically an accident of some kind) [2]. Associated analysis techniques allow the probability of the consequence event to be calculated from the individual probabilities of the events that caused it.

Originally, fault trees were used to build deductive models of how electronic system would behave in the event of various component failures. They have since been adapted for use in accident analysis (see, for example, the discussion in Johnson [9]). In the latter context, they provide a way to reconstruct and express the events that lead up to the accident occurring.

Figure 5.9 gives an example of a fault tree, and a simplified key to the standard fault tree symbols is given in Figure 5.10. The latter is far from complete — see Johnson [9] for more detail — but will be sufficient for the discussion in this section.

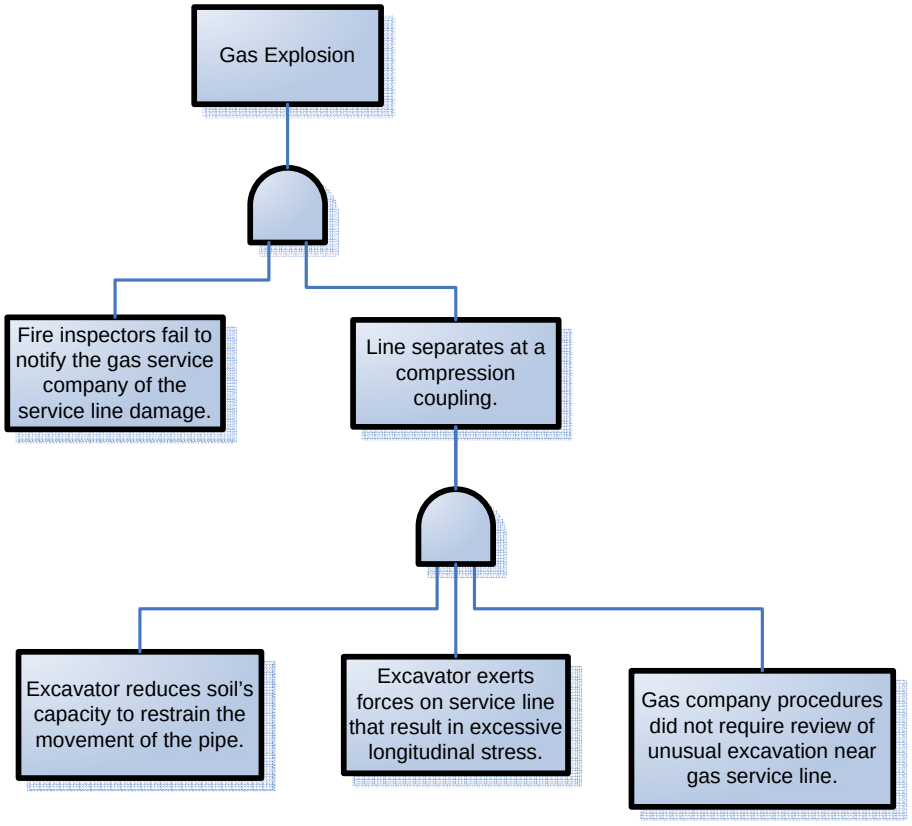

Figure 5.9: An Example of an Accident Fault Tree (reproduced from [9])

For many mechanical or electronic systems, developing a fault tree is a straightforward manual process. For complex SoS, however, creation of adequate fault trees is difficult. The reasons are those that have been noted earlier in this thesis: the behaviour of the overall SoS, including accidents, emerges from complex nonlinear interactions across time, and this is very hard (if not impossible) to determine directly from a static representation (see Section 1.2). In many cases, a given SoS will only provide safety behaviour that is reducible to a fault tree under a very specific set of circumstances, and even then the logical relationships between the causes and the consequence may involve a lengthy sequence of events. An SoS fault tree can be readily constructed, however, using the results of the hazard analysis method presented in this thesis.

As with event trees, it can be observed that the decision trees (and corresponding hazard rules) from Section 5.6.3 provide the information needed to produce a fault tree for an SoS, or at least an SoS in a particular vignette. The deviations used in the rules provide the nodes for a fault tree, the AND and NOT operators provide relationships between those nodes. The existence of rules which provide multiple combinations of deviations that lead to the same accident will give rise to NOT gates. If several rules contain the same deviations (which is likely), each of those deviations should appear only once on the resulting fault tree.

It can be observed, however, that fault trees generated from deviation-hazard rules alone will tend to be 'flat' — they will have basic events (deviations), normal events (vignette context

Figure 5.10: Basic Fault Tree Symbols

introduced in analysis step 6) and the final accident event, but they will not contain intermediate events. Such trees are not tremendously useful because it is not apparent how the presented causes lead to the consequence.

An analyst can use the rules produced by the decision tree learner to produce the skeleton of the fault tree. They can then supplement this using the knowledge gained via the use of agent tracing and other sources (such as visualisation) to compose a satisfactory explanation in fault tree form. They can also restructure the tree to remove oddities resulting from the rules (for example, the initial tree may have duplicate nodes).

There is a tension between the two uses of fault trees (analysis of design and analysis of an accident that has occurred). When analysing the *design*, an analyst will try to find all the events and combinations of events that can lead to the consequent event, thereby supporting a qualitative safety analysis of the system. By contrast, when analysing an *accident* the analyst will seek to describe only those events that occurred in the actual accident scenario. They will exclude all other possible causes (except, possibly, where there is doubt as to which of two possible causes is the correct one). This is relevant here because the simulation analysis leads to a collection of accident analyses, but the overall aim is to build an understanding of the accident behaviour of the whole SoS configuration (effectively the 'design' of the SoS).

A general approach here is to start with the description of each individual single accident rule, using all the information gathered through the analysis steps for that accident, and then progressively combine that with the results of other rules, eventually building a 'design' fault tree which contains several distinct ways in which the accident can occur.

5.11.3 Commentary — Linearization and Simplification

A multi-agent simulation model of an SoS is a complex, nonlinear system in the same way that the SoS itself is. A set of hazard rules, or a fault tree, derived from such a multi-agent model is linear. Richardson, in [126] notes that although such a *linearisation* can be valuable it is inevitably misleading in that it has distorted the dynamics of the original system.

Furthermore, any model derived from the simulation model is a *simplification* and abstraction, and captures only a part of its behaviour. The simulation model in turn is a simplification and abstraction of the actual SoS (if that exists at the time of modelling). A simplified model is invariably partial in terms of time, space, variety of element behaviour, and many other aspects, some of them potentially quite subtle.

The danger is that it is difficult to assess and describe how much linearisation has weakened the predictive power of a model, or how partial a simplified view is. It can be extremely difficult to assess the changes to the system or its context that will invalidate the model and make its predictions unsound. This is, however, the nature of working with complex nonlinear systems that have complex boundaries with their environment — the behaviour of such systems is inevitably hard to predict.

5.11.4 AGO Case Study

Given the explanation from Section 5.7.2 and the generalisation from Section 5.9.1, a fault tree can be produce for the accident ‘gh1’. This is shown in Figure 5.11.

It can be observed that all the events in the tree apart from “gun1 has fire skewed” are ‘normal’ events, which must be treated as having a probability of 1. Although further investigation will be needed (to determine the actual risk of substantial artillery inaccuracy, and to determine the minimum realistic distance between discrete infantry deployments), this implies a high risk.

5.12 Use of the Analysis Results in the SoS Safety Process

As noted in Section 3.7.2, the first role of simulation hazard analysis is in the SoS architecture phase of the SoS Double-V SoS lifecycle (this lifecycle was described). The work presents in this thesis focuses mainly on the SoS architecture phase, but the scope of simulation hazard analysis is, as noted in Section 3.7.2, broader. As described in Section 3.2, this is the stage where SoS-PSSA is performed. The analysis is important here for (a) evaluating the hazards that are likely to occur in the currently proposed SoS architecture and (b) supporting the derivation of safety requirements for the component systems that are to be developed during the system V.

The safety requirements on component systems are of value in all phases of component system development — they provide consequences that are to be avoided (allowing the system owners to perform system PHI and system risk assessment in which hazards that could cause those

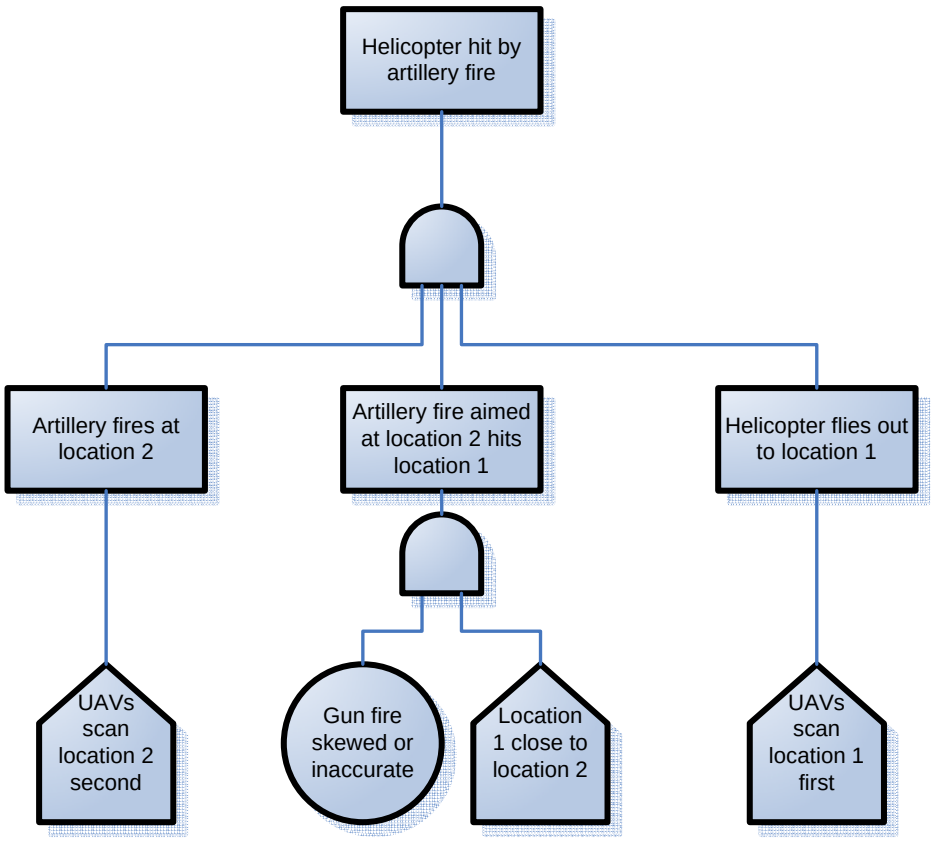

Figure 5.11: Fault Tree for Accident gh1

consequences will be identified), and thereby provide grounds by which component system PSSA and (subject to fidelity limitations) component system SSA can be conducted *in context*, i.e. with knowledge of the effects of the system on the SoS in which it is likely to be embedded. The component system owner is therefore a consumer of the results of the SoS-level analysis (specifically the results of SoS-PSSA). The component system owner may also *produce* the results of exploring the interactions of their component system (possibly modelled more accurately than in the original SoS-PSSA model) with the context that was provided (in the form of the original simulation model) by the SoS owner.

As noted in Section 3.7, this thesis does not propose a method suitable for performing confirmatory safety analysis using a simulation model. The role of simulation hazard analysis in SoS-SSA is therefore limited. However, it is necessary for SoS-SSA to confirm the resolution of all the hazards that were proposed by the simulation analysis, either by revealing them to be implausible or by showing that they have been adequately mitigated.

As discussed in Section 5.8, an analyst must validate the results that are produced from the analysis. Across the length of the safety lifecycle, this must become progressively more thorough and complete. Conducting detailed validation is expensive, and to do so early in the lifecycle (where a great many details are likely to change before the system sees operation) would be

impractical.

An example of this progression could be:

- SoS-PSSA — "face validity" judged by domain experts
- SoS integration — structured reviews with domain experts
- Final SoS-SSA — confirmation using detailed simulation models and live exercises

5.13 Arguing That All Hazards Have Been Identified

It is often necessary, as part of a safety case, to argue that all hazards exhibited by the system have been identified. This is often referred to as a 'backing argument' (as distinct from the primary argument, which concerns the safety of the system). Figures 5.12 through 5.15 show the high-level structure of such an argument in the Goal Structuring Notation (see Kelly in [159] for a description of the notation).

Figure 5.12 shows the use of several complementary hazard analysis techniques as a basis for arguing that all hazards have been identified. The simulation-based hazard analysis approach, as presented in this thesis, is *one* of the techniques, and this supports the overall argument by claiming to reveal hazards that could not be found by the other techniques (see context C2). This is then broken down into separate arguments for the adequacy of the simulation model representation and of the analysis that it is subjected to.

Figure 5.13 shows how the adequacy of the representation could be argued, by asserting that key properties of the model hold, and that the process used to derive it from the source model did not introduce significant errors.

Figure 5.14 shows how the completeness of the set of deviations could be supported, following on from the reasoning presented in Section 4.9.

Figure 5.15 shows how the adequacy of the analysis could be argued by considering the individual methods used to explore the space of possible deviation combinations and to extract hazards from the resulting logs. This is complemented by the claim that the results had been independently reviewed and felt to be plausible.

Working from the framework presented here, the analyst who wanted to contribute to a safety case would need to complete the argument by developing with the undeveloped goals and supplying appropriate evidence. Throughout the thesis we have provided guidance on how many of these goals can begin to be addressed. For example, Section 4.13.1 provides discussion on the necessary output from simulation runs to support analysis that helps address G11 'Model logs sufficient information for analysis'. However, specific arguments must still be created for any specific instantiation of the thesis framework.

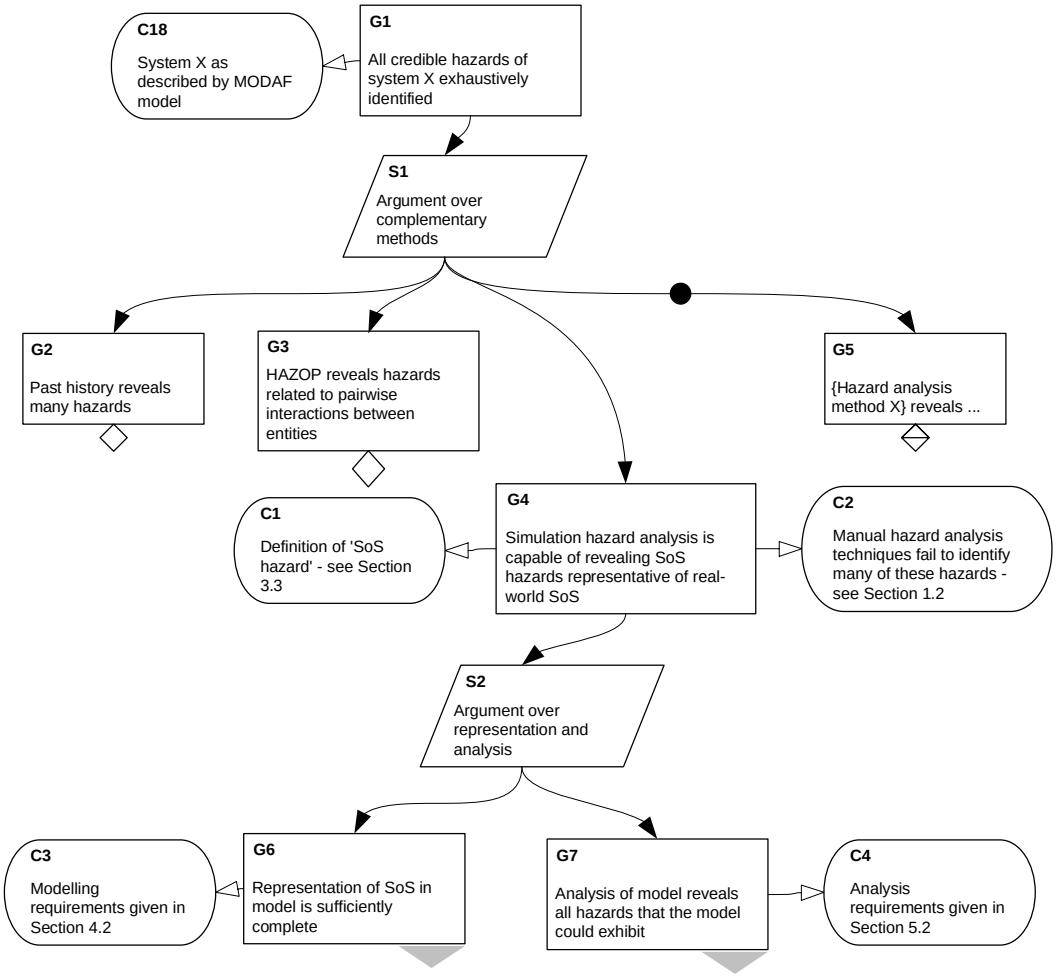

Figure 5.12: Argument That All Hazards Have Been Identified

5.14 Summary

This chapter initially presented a set of requirements for effective hazard analysis of the models produced by the modelling process that was presented in Chapter 4. Using these as a guide, it showed how the space of possible deviation combinations could be explored, thereby revealing the accidents that could occur in the system, and how a decision tree learning algorithm could determine which deviations would cause which accidents. This was then further supported by a description of how an agent tracing tool could be used to explain those relationships in terms of the internal details of the simulation model. Guidance was provided on how the resulting hazard descriptions could be validated, and then presented to other engineers using fault trees. All aspects of the analysis process were illustrated using the ongoing case study.

In the next chapter the process described in Chapters 3 through 5 is applied to an additional realistic case study.

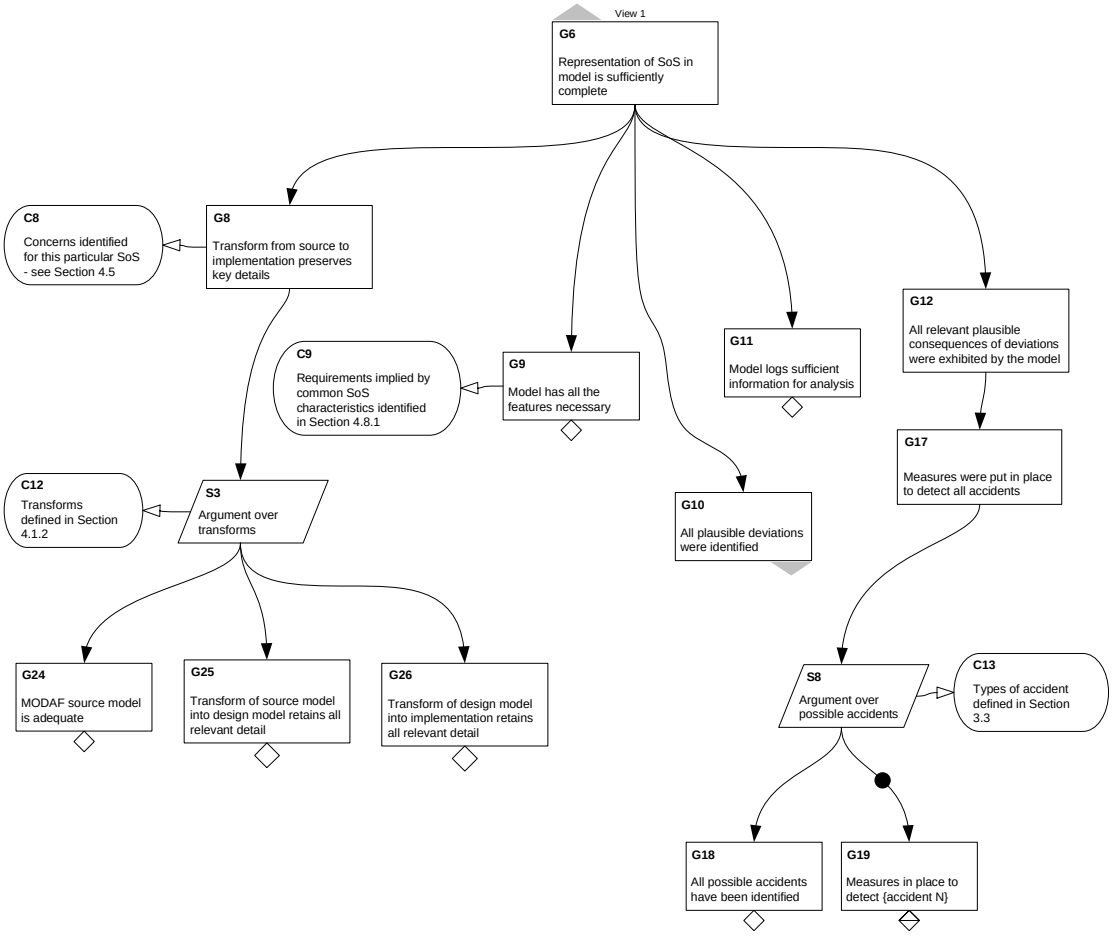

Figure 5.13: Argument That the Representation of the SoS is Adequate

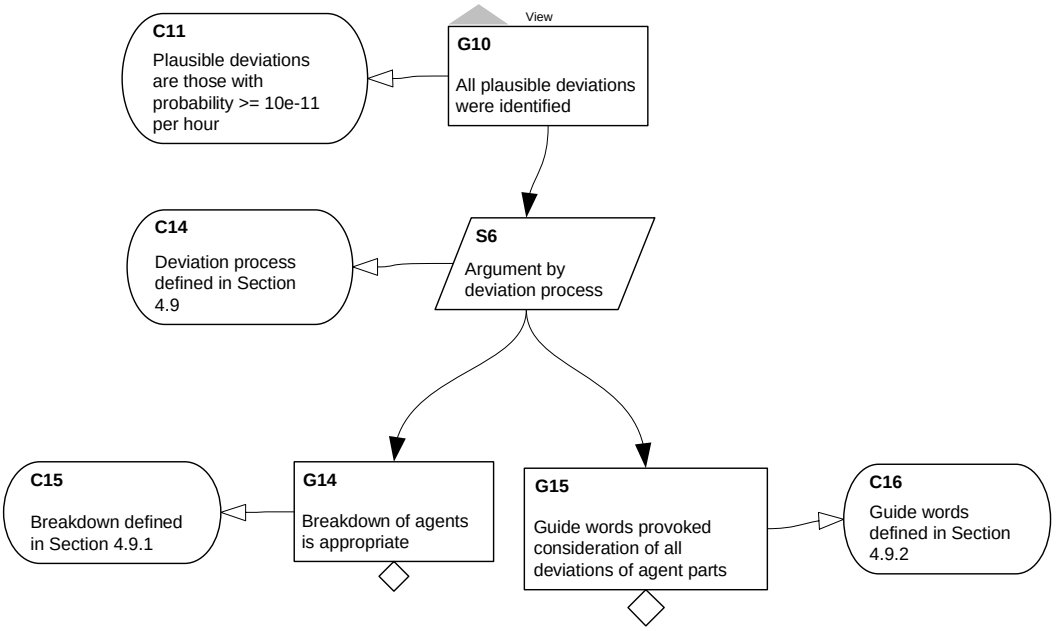

Figure 5.14: Argument That All Important and Plausible Deviations Have Been Identified

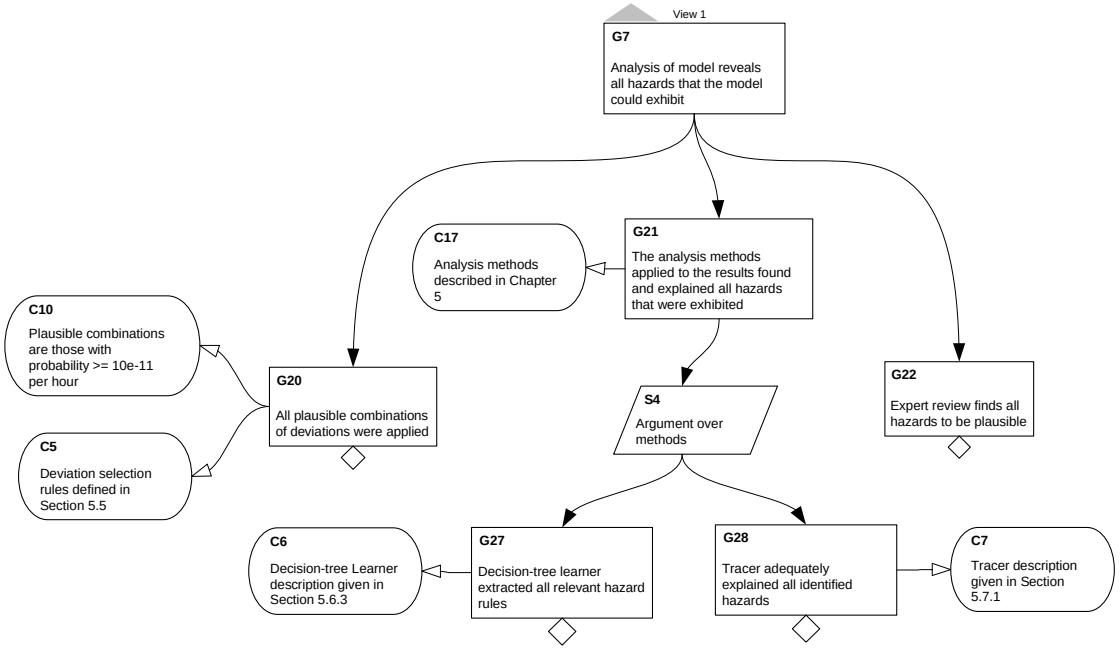

Figure 5.15: Argument That the Analysis of the Model is Adequate

Chapter 6

Case Study

The three preceding chapters have presented an approach to SoS hazard analysis. In this chapter, a case study will be performed to demonstrate the application of the approach in practice and to support the evaluation in Chapter 7.

Examples from the modelling and analysis of the 'AGO' system and scenario were presented in previous chapters. A new system and scenario will be used here, and taken through all the steps of the modelling and analysis process.

6.1 Adcock Case Study

The system described here is will be known as the 'Adcock' system. It consists of a small military unit where a central command entity manages a set of UAVs that provide surveillance and reconnaissance over a large area. This information is then used to support the action of ground-based infantry and special forces. The activity of the various elements in the Adcock system is coordinated via a Combined Operational Picture (COP) created primarily by the fusion of data from sensors mounted on the UAVs and other entities, (although the COP could potentially incorporate data from other sources, such as plans and conventional intelligence). A graphical representation of the Adcock system can be found in Figure 6.1 in the form of a MODAF OV-1a (High Level Operational Graphic), although it can be noted that this includes a variety of force elements, such as naval assets, that will not be considered in this case study.

The Adcock system is described by Adcock in [10] and [119] and has been used in various projects including the BAE Systems EPSRC-funded NECTISE research project. Although not a currently deployed military system, it can be considered representative of such.

This case study will run through all process steps in a single iteration of the hazard analysis process, as described in Figure 3.3. It should be remembered that in a real industrial context this iteration would be one of many. Some observations will be made in the text about how later iterations would be likely to proceed. Consistent with the rest of the thesis, the case study will concentrate on process phases 1 and 2, with phases 3 and 4 being mostly out of scope.

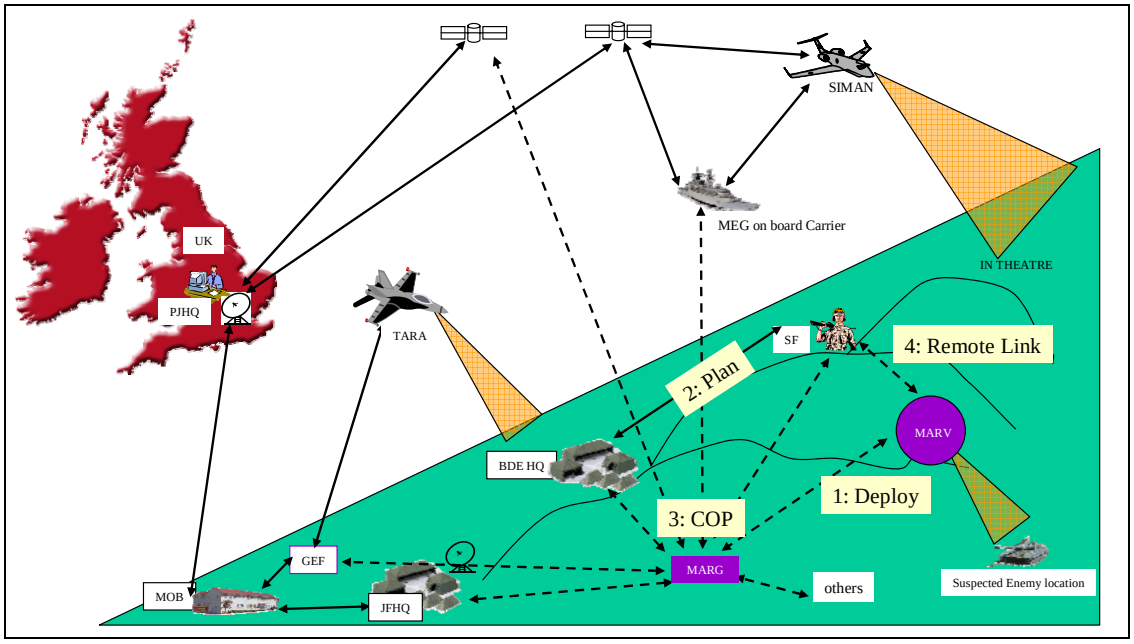

Figure 6.1: OV-1a for Adcock System (from [10])

This chapter presents the results of following the process as described in Chapters 4 and 5. Representative extracts of the models created will be used throughout the chapter. The complete set of artefacts created can be found in Appendices C and D.

6.2 Phase 1 — Build Model for Adcock Study

6.2.1 Step 1 — Acquire MODAF Model

A partial MODAF source model was available for this case study. It describes a system where a UAV (known as the 'MARV') performs reconnaissance which was used by other system elements to support the formation of a Combined Operational Picture (COP). The MARV is controlled by a ground station (the 'MARG'). The MODAF model describes how a special forces unit would call for reconnaissance and how the rest of the system would coordinate to provide the requested information.

This is a valuable starting point, but it is not sufficient for safety analysis — there are no effectors (such as armed platforms) that could cause serious damage. The decision was therefore made to model an expanded version of the system that featured an additional armed UCAV (Unmanned Combat Air Vehicle) that would provide air support for the ground forces.

6.2.2 Step 2 — Identify Concerns

The concerns identified for the case study, and their corresponding representation in the Adcock MODAF model, are shown in Table 6.1.

6.2.3 Step 3 — Identify Missing Information

Since new entities were introduced in step 1 that were not present in the original source model, additional information is needed about their likely behaviour, and concerning how the UCAV would be integrated into the system.

In a real industrial situation it would be appropriate to call a meeting with domain experts at this point, in order to get an idea of how the integration of a UCAV would be most likely to happen in practice. For the purposes of this case study, plausible assumptions were made.

As a consequence of this information, a revised OV-6c sequence diagram was produced, and this is shown in Figure 6.2.

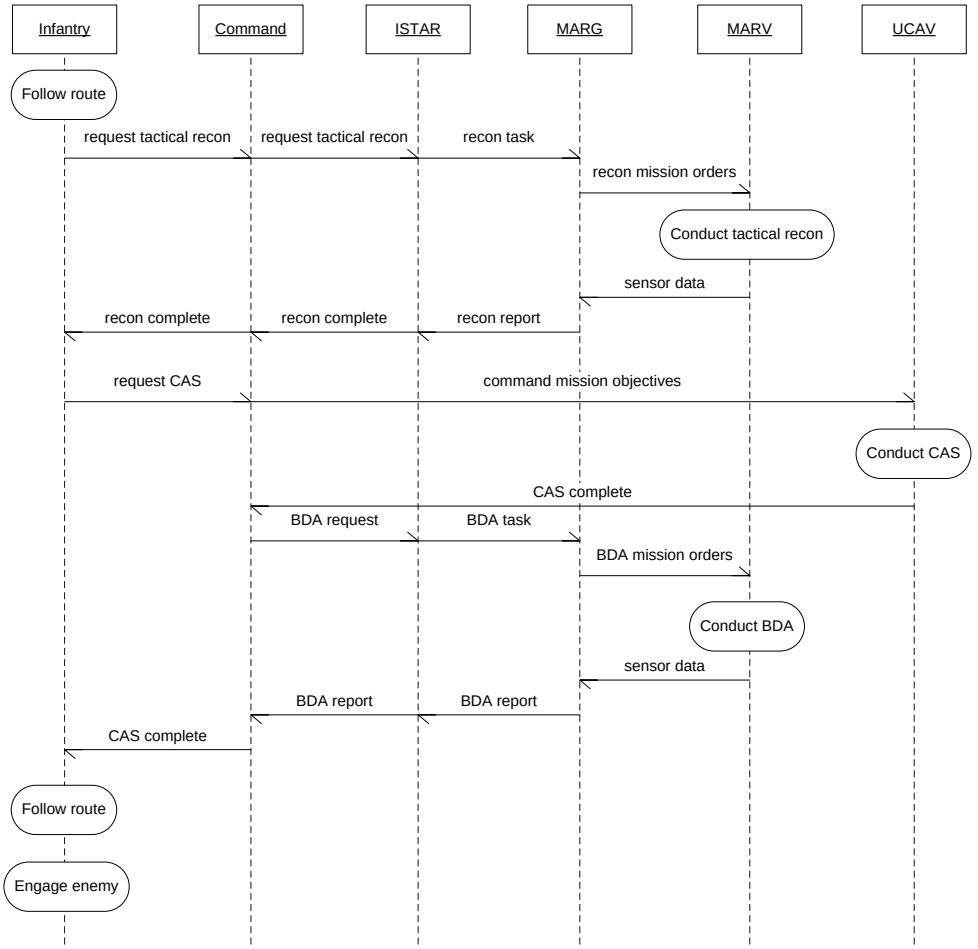

Figure 6.2: Revised Version of OV-6c

ID	Type	Description	Model Significance	MODAF Significance
1	Accident	Enemy infantry destroys infantry	Enemy position at different times, infantry position and movement, relative effectiveness of infantry weapons, armour and tactics versus enemy	Not represented
2	Accident	Enemy tank destroys infantry	Tank position at different times, infantry position and movement, relative effectiveness of infantry weapons, armour and tactics versus tanks	Not represented
3	Accident	UCAV hit infantry with air strike	Infantry position at different times, location and timing of air strikes, effect of air strikes on infantry	Not represented
4	Accident	UCAV collides with MARV	Position of UCAV at different times, position of MARV at different times	Not represented
5	Deviation	Limited radio communications bandwith	Availability of bandwidth, consumption of bandwidth per message, message sending by time	Needlines are in OV-2b, communications are implicit in OV-5, OV-6. System-level communications are in SV1 and SV2, and forecast for communication technologies is given in SV9.
6	Deviation	Failure to include some/all entities in situational awareness	Individual entity SA state over time, timing and content of sensor information and SA update messages	OV-1b notes that a COP is created, OV-2b shows distribution of reconnaissance data and COP, OV-5 and OV-6 show reconnaissance data messages.
7	Deviation	Errors in coordinate system or landmark references	Mapping mechanism between world coordinates and entity SA coordinates	Not represented
8	Deviation	Errors of coordination and timing between the infantry and the rest of the system	Time and location of infantry actions, the coordination protocols that are used	The scenario described in OV-5 and OV-6c shows the expected order of action — OV-5 shows 'flow of control', OV-6c shows the expected sequence of interactions.

Table 6.1: Adcock Concerns Against MODAF

The new OV-6c shows an infantry unit moving on a predefined route. It suspects the presence of an enemy unit nearby, so it contacts command and asks for local reconnaissance to be performed. The command node arranges (via the ISTAR[1] coordination node and the MARG) for the MARV to perform this reconnaissance. Once this is completed, the MARV sends the sensor data back to the MARG, which interprets it, fuses it with the COP, and sends a 'reconnaissance complete' report back up the chain to the infantry.

Studying the updated COP, the infantry unit observes an enemy unit nearby. The enemy have a tank with them, and the infantry unit is not equipped to deal with this. The infantry therefore calls for CAS (Combat Air Support) to deliver an air strike on the enemy location. The command node arranges for the UCAV to perform this, and once the UCAV reports completion of its mission the command node arranges BDA (Battle Damage Assessment) via ISTAR in the same way as it requested reconnaissance. The BDA report shows that the tank has been destroyed, so the command node now tells the infantry that CAS is complete. The infantry then continues moving along its route, and may safely engage the enemy.

6.2.4 Step 4 — Decompose Scenarios into Vignettes

The source model for this case study describes the scenario in terms of options for missions, environments and threats, giving several possibilities (not all mutually exclusive) for each. This can be combined with vignette aspects derived from the identified concerns to provide a large number of potential vignettes. The concerns (see Table 6.1) were used to derive the aspects shown in Table 6.2.

Concern	Aspect
1	Enemy troops present
2	Enemy tank(s) present
3	Tanks (or other air strike targets) in areas where infantry will go
4	MARV occupies space that UCAVs will want to fly through
5	SoS shares bandwidth with large number of other allied entities
6	Additional friendly entities (not part of SoS) in area
7	No implications identified
8	No implications identified

Table 6.2: Vignette Aspects Derived From Concerns

In this case study, we will focus on one vignette. This vignette will incorporate the first four aspects identified above.

[1]ISTAR is *"Intelligence, Surveillance, Target Acquisition and Reconnaissance"*

The initial situation for the vignette will be:

The system units are located in a region of desert terrain. There are three infantry units served by a single MARV and UCAV (with associated command and ground station entities). A set of three enemy units are located within the area of concern. Waypoints have been assigned to the infantry that will cause them to cross the area of concern, bringing the whole area under the sweep of their ability to detect enemy presence.

The sequence for the vignette is the same as that shown for the scenario in Figure 6.2.

The expected final situation for the vignette is:

All system entities are still present and operational. The infantry have finished their sweep routes and are located at the far side of the area. The MARV and UCAV are loitering in their original positions. All enemy units have been destroyed or disabled.

6.2.5 Step 5 — Transform Domain Model into Design Model

As noted in Section 4.8.2, the design model defines a fixed set of component systems who are to be modelled as agents in the simulation. Any decisions about which MODAF nodes are to become agents must therefore be made at this stage. In the current case study, the decision was made to combine the several command and control nodes (including the remotely located 'home' command centre) depicted in the various MODAF artefacts into a single command entity. The various surveillance-coordination nodes were likewise combined into a single 'ISTAR' entity.

These decisions were forced by lack of information — the remote command, for example, did not feature in OV-5 or any of the other behavioural descriptions. They did have the advantage of allowing the model to focus on those entities that were physically mobile and active in the field.

Once the above decisions were made, the Prometheus process was followed to a produce the design model that is given in Appendix C. The system overview diagram is reproduced in Figure 6.3. A key to the notation was provided in Figure 4.4.

6.2.5.1 Architectural Design Cross-Check

After the architectural design phase of Prometheus was complete, the resulting artefacts could be cross-checked against the MODAF as described in Section 4.8.3.

The System Overview diagram was consistent with product OV-1, given knowledge of the changes that were explicitly made in earlier steps — some elements (e.g. the UCAV) were added to expand the scenario, some elements (e.g. remote command) were lost due to merging of entities. Some elements that had not been previously excluded were not represented in the Prometheus model, such as the manned aircraft shown in OV-1. This was largely because none of the other MODAF products referred to them, and hence they could not usefully be modelled.

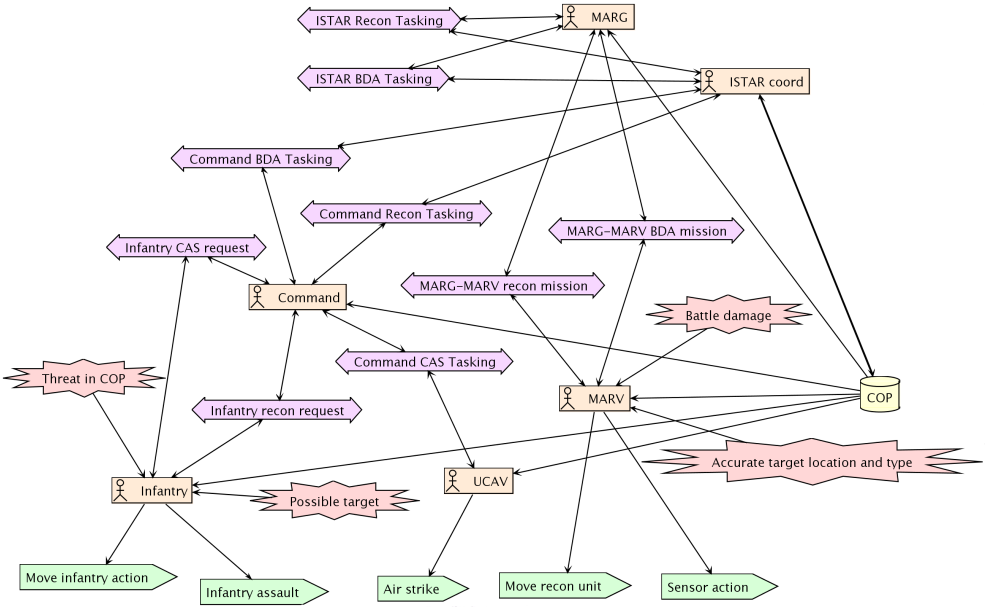

Figure 6.3: System Overview Diagram for Adcock System

The protocols from the system overview were determined to be sufficient to give rise to all the interactions described in OV-5 and OV-6c (although it can be noted that the 'data analysis' node from the original OV-6c has been merged into the MARG).

All protocols occur over OV-2 data links (given that "carrier" and "joint" ISTAR are merged), with the exception, of course, of communications with new entities not shown in OV-2. The protocols do extend beyond the communications described in OV-3, but the OV-3 here is visibly incomplete; for example, no information appears to be given to the ground forces.

At this stage a cross-check of the Prometheus message descriptors with OV-7 would have been performed, but, although OV-7 was present in the MODAF, none of the things it contained could be related to any of the nodes or interactions in OV-5 or OV-6c. Furthermore, it contained strange entities (such as apparently *physical* objects). For the purposes of this case study, OV-7 was ignored. In a real industrial application, clarification would be sought at this point from the developers of the source model.

6.2.5.2 Detailed Design Cross-Check

The source model provides only a very simple example of OV-6b, and this applied only to the MARV node. The design model is, however, broadly consistent with this. The exception is the 'fault' state leading to a 'maintenance and training' state. As this refers to a longer timescale than that of the analysis that will be performed here, and because fault states will be devised in the next step of the method, it is acceptable to ignore this here.

6.2.5.3 Tracing Concerns

The tracing of the top-level concerns to their support in the design model is shown in Table 6.3. It can be observed that a number of the concerns are not represented in the model at this stage. This is not a problem, provided that they are reintroduced later. The tracing of concerns after each model development stage helps to ensure that this reintroduction occurs.

Concern	Design Elements that Address Concern
1	Not represented
2	Not represented
3	UCAV has 'air strike' action. Target coords supplied via an explicit message
4	Not represented
5	Most data exchange is via explicit messages (only exception is COP, which can be converted into explicit messaging using a single mechanism)
6	COP explicitly represented. SA (situational awareness) data enters explicitly through percepts. Reconnaissance data is explicitly passed to ISTAR before being integrated into COP (only ISTAR 'writes' to COP).
7	Not represented
8	Scenario captures the normal/expected case. Coordination emerges from explicit individual actions and is achieved by explicit messaging.

Table 6.3: Adcock Concerns Against Design Model

We can conclude that design model is adequate given the limitations of the source model. It captures the behaviour of a subset of the entities in the model for which sufficient detail was provided in the source model.

For the purpose of this case study, the design model will be used as is. In a real application of this method, it is likely that further clarification would be sought at this point to clarify that the model built here was adequately representative of the proposed SoS.

6.2.6 Step 6 — Deviate Design-Level Agents

Once the design model was apparently complete, deviations were derived as described in Section 4.9. The resulting deviation table can be found in Appendix D. Some examples of the deviations described in the table are presented below:

- All entity types may have their communications impaired. They may lose the ability to send messages or to receive them. The bandwidth required to send each message may be increased, or a delay may be introduced prior to the actual transmission of a sent message.

- Similarly, all entities may have their situational awareness impaired. The coordinate system they use may be skewed (so that the stored coordinates for a sensed entity will not match its actual position in ground truth) or the number of entities that can be maintained in awareness may be limited (potentially causing entities to be lost from memory).

- Infantry entities may have their sensor range reduced (or increased), and they may have a delay introduced prior to any attack action. Deviations to their plans are also suggested — for the 'Find Possible Target' plan (which is triggered when a percept is received suggesting that unknown entity is nearby), deviations include requesting reconnaissance twice, and continuing to move without waiting for the reconnaissance results.

For each deviation, a probability of occurrence can be calculated. As an example, consider the 'radio cannot send' deviation that is common to all entities. If the allied entities in the scenario use a common radio technology, they might be assigned a common probability of this deviation based on historical failure rates for that implementation. Let us assume that this figure is 1×10^{-4} per operating hour, which is used as the probability per vignette instance.

If this were to be combined with another deviation where the existence of a common cause was suspected, a combined probability could be calculated using the formulae in Section 4.9.5.2. For example, if this deviation was 'Situational Awareness does not persist' on the same entity and this had an independent probability of 1×10^{-3} then a BETA factor could be derived of $1.0 \cdot 0.1 = 0.1$, leading to a common-cause deviation probability of $0.1 \times \frac{10^{-4}+10^{-3}}{2} = 5.5 \times 10^{-5}$.

Similarly, if 'radio cannot send' were to be combined with 'radio cannot receive' then this would be a pair of deviations of the same component on the same entity, leading to a BETA factor of 1 and hence a combined probability of $1 \cdot 10^{-4} = 10^{-4}$

As noted in Section 4.9.5.2, applying the Partial BETA technique indiscriminately, without consideration of whether there are grounds to suspect a common cause, will lead to unreasonably high probabilities of deviations occurring together, which will distort the results of the analysis.

For simplicity of modelling and analysis in this case study, a uniform probability of 1×10^{-3} will be used for all deviations. Given the difficulties in implementing the Partial BETA method, all deviations will be assumed to be independent.

6.2.7 Step 7 — Deviate Design-Level Vignettes

For this case study, only one vignette-level deviation was defined: "reduced bandwidth available". This could represent, for example, having to share the available spectrum with other allied forces, without the complexity of explicitly simulating those allied entities and the traffic that they would generate. It can be noted that this goes some way to addressing concern 5, which was not particularly addressed by the baseline version of the vignette we are using.

6.2.8 Step 8 — Trace Concerns to Deviations

The mapping of the concerns to the defined deviations is shown in Table 6.4. It can be seen that all deviation concerns have been addressed, but that this stage has not improved the representation of accident concerns.

Concern	Deviations
1–4	(Accidents, obviously, are not directly represented in deviations)
5	All agents can have their bandwidth use per message increased. The bandwidth available to the whole vignette can be reduced.
6	All agents can have the maximum number of entities of which they are aware restricted to a small number. Entities with organic sensors can have their sensor range reduced, and a delay introduced between them sensing something and registering it in SA.
7	All entities can have their coordinate system skewed for the purposes of SA.
8	The infantry can have their movement speed doubled, or their plans changed so that they don't wait after calling for reconnaissance or air support. The MARV can have a delay introduced in the planning of reconnaissance missions.

Table 6.4: Adcock Concerns Against Deviations

6.2.9 Step 9 — Transform Design to Operational Model

The case study was implemented in the Sim8 simulation engine, as described in Section 4.12.4. Sim8 was adopted for this study as it was available, computationally efficient, and sufficient in terms of features. Naturally, the adoption of this platform had significance for the operational model. The mapping of concerns to operational model features is shown in Table 6.5. It can be observed that although some concerns were not represented at the design model stage, they have all been addressed in the operational model.

It can be noted that at this stage, as there was no apparent benefit to ISTAR and Command being kept as separate entities (since neither of them were represented in space, and much of the traffic with ISTAR went through Command anyway), both were merged into the Command agent.

6.2.10 Step 10 — Implement Operational Model

Once the model had been modified to conform to the operational model imposed by Sim8, it was implemented. This involved approximately 2500 lines of Java code. Table 6.6 shows how

Concern	Support in Sim8
1	Supports small arms fire
2	Detects potential collisions
3	Support for indirect/area fire
4	(as 3)
5	Engine supports limited bandwidth. Messages are delayed if insufficient bandwidth. Bandwidth costs can be varied.
6	Supports simple sensors, simple SA.
7	No explicit support, but agents use their own coordinate systems internally so can distort it if desired.
8	Agent behaviours must be specified such that coordination occurs — there is no central mechanism that allows coordination to be defined. Likewise, emergent properties that relate to timing can be observed.

Table 6.5: Adcock Concerns Against Operational Model

the concerns map to logging output of the implemented simulation model. It can be seen that all the concerns have will have some representation in the logs that are output.

Concern	Log Output
1	All attacks, all agents destroyed
2	All collisions
3	(as 1), plus attack goals and plans formed by friendly entities
4	(as 1)
5	Delayed messages logged (at point of both sending and delivery)
6	Beliefs about entity location
7	Beliefs, goals and plans contain agent-internal coordinates. External actions contain world (ground truth) coordinates.
8	The log events are stored in chronological order (at the sub-tick level)

Table 6.6: Adcock Concerns Against Log Output

The normal-case (undeviated) log output can be filtered and processed so as to produce a sequence of events that can be compared to the normal-case sequence diagram given in Figure 6.2.

There are some complications here. First, the domain model assumes a single infantry unit and a single enemy unit, whereas the implemented model assumes multiples of both. This is certainly a reasonable difference, but it complicates the comparison. The domain-model sequence diagram also seems to assume that fusion of sensor information only happens as a

result of occasional, deliberate exchanges between agents, whereas in the model it happens continuously and automatically. This latter difference is plausible, and the text of the source model is consistent with a continuous fusion approach. A similar issue was noted for the AGO model in Section 4.4.1.

The two differences are reasonable, so for the purposes of the case study they will be accepted. As ever, in a real application it would be advisable to seek clarification at this point.

Appendix E contains an extract from the log output produced by the undeviated version of the Adcock vignette. It has been edited to exclude the actions of the additional infantry agents, although their effects can be observed, for example, in the delays between infantry reconnaissance requests and actual MARV tasking.

It can be observed that the log maps across to the sequence diagram shown in Figure 6.2:

- At time tick 0, inf1 has an assigned waypoint (which will lead it to follow a route to that waypoint).
- At time tick 10, inf1 observes movement nearby and calls for recon from Command, which requests reconnaissance from MARG.
- At time tick 11, MARG tasks the MARV to perform the reconnaissance.
- From time tick 11 to time tick 74, the MARV carries out its reconnaissance mission.
- At time tick 67 the MARV detects an enemy position and sends this information back to the MARG, which propagates it to Command which sends it to inf1.
- inf1 responds, at time tick 71, by requesting CAS from Command, and in response Command tasks the UCAV.
- At time tick 104 the UCAV delivers an air strike at the target location and sends a message to Command indicating CAS completion.
- Command immediately responds by ordering the MARG to arrange BDA.
- The MARG then tasks the MARV to perform BDA, at time tick 105.
- At time tick 147, the MARV starts to conduct BDA.
- At time tick 192, the MARV indicates BDA completion.
- The MARG propagates this completion to Command at time tick 193, and Command then indicates to inf1 that CAS is complete.
- From time tick 194 onwards, inf1 moves and attacks the enemy.

It can be observed that the continuous data fusion that was actually implemented causes a change in the order of messages and actions — in the MODAF model the infantry requests

CAS only after the MARG indicates that the entire reconnaissance mission is complete, while in the simulation the infantry requests CAS very soon after the MARV observes the presence of the tank. This is consistent with the earlier observation that MODAF models do not provide a clear way to express continuous data fusion and autonomous behaviour based on the result of that. The discrepancy is therefore acceptable.

6.2.10.1 Implementation of Deviations

A subset of the possible deviations identified for the system in Appendix D were implemented. Table 6.7 shows a list of these, along with a set of more descriptive identifiers that were used in the implementation.

ID	Code	Description
A1	RADIO_CANT_SEND	The entity can't send radio messages
A2	RADIO_CANT_RECEIVE	The entity can't receive radio messages
A6	RADIO_EXTRA_BANDWIDTH	All radio messages sent by the entity consume extra bandwith
A7	RADIO_SEND_DELAY	All radio messages sent by the entity are delayed
A12	MAX_SA_ENT_2	The entity only remembers the details of the last two entities it sensed (or was told about)
A14	COORD_SKEW_N	When storing the location of another entity, the entity shifts its location 1 square to the north
IN16	INF_DOUBLE_SPEED	The infantry unit moves at twice its standard speed
IN28	INF_FIND_TARGET_DONT_WAIT	After detecting a potential target, the infantry unit continues with its mission without waiting for reconnaissance results
IN38	INF_NO_CAS_LOCK	After calling for CAS, the infantry unit moves to engage the target
MG6	MARG_PLAN_RECON_TIME	The time taken for the MARG to plan a reconnaissance mission for the MARV is increased from 2 ticks to 16 ticks

Table 6.7: List of Deviations Implemented for the Case Study

6.3 Phase 2 — Analyse Adcock Model

6.3.1 Step 1 — Acquire Model

As the modelling process described in this thesis has been followed, this step is not necessary.

6.3.2 Step 2 — Explore the Space

As noted in Section 6.2.6, all deviations were treated as having a uniform probability of occurrence, which was set at 1×10^{-3}. A minimum run probability was defined at 1×10^{-11} per run (based on the 'incredibility of failure' concept as explained in Section 5.5).

Consistent with the above minimum probability, all runs were performed with up to four simultaneous deviations. This lead to a total of 294204 runs. The mission completed in 9843 runs, failed in 34650, and timed out in 249711. Execution took 18969 secs (a little over 5 hours), and the automated part of the analysis took 4041 secs (a little over an hour).

6.3.3 Step 3 — Learn Hazard Rules

Log events corresponding to accidents were identified using regular expressions, allowing input data for the decision tree learner to be generated. Rules were then learned for each accident that was exhibited in the run set. The three most probable rules for each accident are shown in Table 6.8. (As described in Section 5.6.3.6 the negative terms are treated as having a probability of 1, so their presence doesn't affect the overall probability of each rule.)

As described in Section 5.6.3.6, the severity of each accident can be defined, and then combined with the probability to produce a measure of risk. The accidents exhibited by the system fall into two main categories: loss of infantry (due to enemy action or friendly fire) and loss of UAVs (due to collision). Using a MIL-STD 882D severity table, as was given in Table 5.2, both types of accident can be considered *catastrophic* as they involve either death or substantial material loss.

Taking the accident cM1U1 (a collision between the MARV and the UCAV) as an example, we have a catastrophic accident where the highest probability rule is 1×10^{-6} per run. We will make an assumption, for purposes of illustration, that the average run is of one hour in duration, allowing us to use this as the probability *per hour*. According to the frequency classifications given in Table 5.3, 1×10^{-6} is a 'remote' probability. The risk matrix in Table 5.4 indicates that a catastrophic accident of remote frequency is of category 8, which falls in the 'serious' risk band.

A 'serious' risk classification may be acceptable, depending on the context. There are, however, many sources of uncertainty in this probability, ranging from the probability of the relevant deviations through the likely position of enemy units on the ground to the precise details of the UAV's flight behaviour. It is quite possible that the actual probability of this accident is 1×10^{-5}, which is on the boundary of the 'occasional' frequency region. A catastrophic, occasional accident is in risk category 4 on Table 5.4, and therefore in the 'high' risk band. This is unlikely to be acceptable.

Alternatively, if specific figures for the likely monetary cost and loss of life associated with each accident were supplied, a quantitative approach to severity could be used. The expected monetary value and expected loss of life could then be summed across all accident rules, providing

Accident	Cases	Rules	Probability
cm1u1	421	'deviation_radio_send_delay_inf1', 'failure_radio_cant_send_inf2'	1×10^{-6}
		'deviation_radio_send_delay_inf1', '!failure_radio_cant_send_inf2', 'deviation_radio_send_delay_marv1', 'deviation_coord_skew_n1_command'	1×10^{-9}
		'!deviation_radio_send_delay_inf1', 'deviation_radio_send_delay_marv1', 'deviation_inf_dont_wait_for_cas_inf3', 'deviation_coord_skew_n1_command'	1×10^{-9}
ei1	2	-	0
ei2	1	-	0
ei3	686	'failure_radio_cant_receive_inf1', 'deviation_max_sa_ent_inf3'	1×10^{-6}
		‘!failure_radio_cant_receive_inf1', 'deviation_inf_dont_wait_for_cas_inf3', 'deviation_inf_double_speed_inf3'	1×10^{-6}
		'failure_radio_cant_receive_inf1', '!deviation_max_sa_ent_inf3', 'deviation_coord_skew_n1_inf3', 'deviation_radio_send_delay_command'	1×10^{-9}
ti1	34648	'deviation_inf_find_target_dont_wait_inf1'	1×10^{-3}
		'!deviation_inf_find_target_dont_wait_inf1', 'deviation_inf_dont_wait_for_cas_inf1',	1×10^{-3}
		'!deviation_inf_find_target_dont_wait_inf1', '!deviation_inf_dont_wait_for_cas_inf1', 'deviation_max_sa_ent_inf1', 'deviation_max_sa_ent_command', 'deviation_coord_skew_n1_command'	1×10^{-9}
ti2	19461	'deviation_inf_dont_wait_for_cas_inf2'	1×10^{-3}
		'!deviation_inf_dont_wait_for_cas_inf2', 'deviation_inf_find_target_dont_wait_inf2',	1×10^{-3}
		'deviation_inf_dont_wait_for_cas_inf2', 'deviation_inf_find_target_dont_wait_inf3'	1×10^{-6}
ti3	15148	'deviation_inf_find_target_dont_wait_inf3'	1×10^{-3}
		'!deviation_inf_find_target_dont_wait_inf3', 'deviation_inf_dont_wait_for_cas_inf3', 'failure_radio_cant_receive_ucav1'	1×10^{-6}
		'deviation_inf_dont_wait_for_cas_inf3', 'failure_radio_cant_receive_inf1'	1×10^{-6}
u1i2	2067	failure_radio_cant_send_inf3	1×10^{-3}
		'failure_radio_cant_send_inf3', 'deviation_radio_send_delay_inf1'	1×10^{-6}
		'failure_radio_cant_send_inf3', 'deviation_radio_send_delay_command'	1×10^{-6}
u1i3	7	-	0

Table 6.8: Top Hazard Rules for Adcock Case Study

a tentative assessment of the overall safety risk presented by the SoS.

6.3.4 Step 4 — Explain Hazard Rules

In this section, and those following, the focus will be on the most interesting rules from an illustrative perspective, rather than, necessarily, the most severe.

6.3.4.1 UC1I2 — UCAV Destroys Infantry2

In this accident, the UCAV delivers an air strike to the location where Infantry2 is located (grid square (36,9)). In the sequence defined in the domain model for the normal-case vignette, the air strike is shown as occurring *before* the infantry move on to the target location — they should not, therefore, reach the target location before it occurs.

In Table 6.8, the highest-probability rule for this accident is the single deviation 'failure_radio_cant_send_inf3'.

Initially, the tracing tool displayed only the trace shown in Figure 6.4. This explains that the UCAV performed the air strike because it had finished its previous objective (fly to a waypoint) and had an active plan to carry out an air strike at that location, but offers no explanation as for why the UCAV had that plan active, or why the infantry unit was located there when the air strike landed.

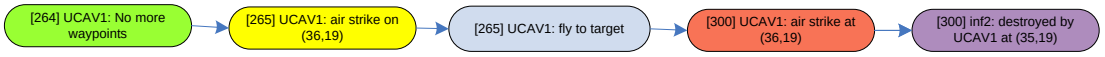

Figure 6.4: Initial Trace for UC1I2

Additional logging was added so that individual (single grid square) moves were tracked, and a tracing rule defined that would relate them to the previous 'move to location' action. This then allowed a rule to be written that could explain air strike casualties in terms of the movement of the recipient as well as the actions of the attacker:

```
An air strike death is caused by an air strike iff:
  The air strike is at the location of the death
  AND the air strike was performed by the agent who caused the death
  AND the air strike is the most recent air strike at the location

An air strike death is caused by a move action iff:
  AND the move action was the last one performed by the dead agent
  AND the move action took the agent into the location that it died at
```

The resulting trace, shown in Figure 6.5, shows the infantry unit is moving to attack enemy3, who was located at (36,9) i.e. the location that was hit by the air strike. It would follow that the UCAV was attempting to destroy enemy3.

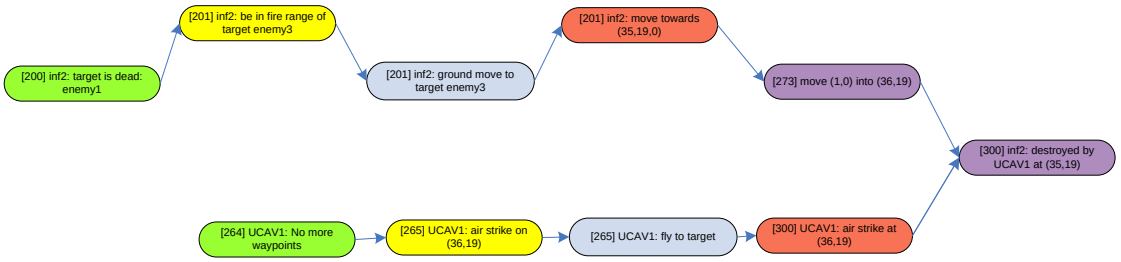

Figure 6.5: Improved Trace for UC1I2

A manual inspection of the log reveals the precise timing of the accident sequence, with the order of air strikes and arrivals at the location. The timing is such that Infantry2 arrives near (36,9), calls down an air strike on it, and, after BDA is performed by the MARV, it moves on to enter the location and destroy enemy3. While this is happening, Infantry1 also arrives near (36,9) and also orders an air strike. It is this second strike that hits Infantry2.

It can also be observed from the log that Infantry2 waits at (36,9) for an extended period of time prior to the air strike arriving. Inspection of the *normal-case* log reveals that Infantry2 moves on almost immediately after destroying enemy3. If that occurred in this case, the accident would be avoided.

The difficulty here for the tracing tool is that, by its nature, a tracing tool as described in Section 5.7.1 cannot have 'unevents' — a causal node cannot be of the form 'event X did not occur', which is what is needed to explain this hazard (e.g. 'Infantry2 does not move away'). However, a manual inspection of the accident log reveals that after arriving at the final location Infantry2 calls for MARV reconnaissance on Infantry3 — it is calling for airborne reconnaissance of a *friendly* entity.

An explanation for this can be found in that the infantry units only call reconnaissance for entities which they have not previously seen, either with their organic sensing or through the COP. Normally, all infantry units appear in the COP soon after the start of the run, because they call out the MARV to perform reconnaissance and it observes them as it does so. However, because Infantry3 has here lost its ability to send radio messages, it has been unable to call for any reconnaissance and so has not be seen by the MARV. It is therefore not in the COP, and hence Infantry2 sees it as an unknown entity.

It can be noted that the location of Infantry2 is in the COP, so potentially the air strike could have been cancelled because it would inevitably lead to blue casualties. However, the behaviour of the UCAV defined in the model does not include any such check.

6.3.4.2 TI1 — Tank Destroys Infantry1

In this accident, the blue infantry moves into fire range of a red tank, and is quickly destroyed. The sequence of operations, with MARV reconnaissance and the option of UCAV air support, is

meant to prevent this, but in this case it does not. When analysed in tracer, no useful explanation is provided — indeed the causal trees look no different from the normal case.

An inspection of the log or the visualisation reveals that the infantry just continues to move after calling for reconnaissance, without waiting for reconnaissance to complete. The infantry is therefore unaware of the tank until it is too close. This is a direct effect of the deviation.

The tracing tool can help little here because of the same problem that occurs in the later stages of explaining UC1I2 — the tracing tool cannot provide explanations in terms of 'unevents' (e.g. "infantry did not wait for recon").

6.3.4.3 cM1UC1 — MARV Collides with UCAV

In the first example found for this accident, a mid-air collision occurs between the two agents at location (31,9). The tracing tool provides the graph shown in Figure 6.6 — this shows that the two agents collided because they moved into the same square, and that those moves were in turn caused by their plans — the UCAV was heading out to perform an air strike while the MARV was returning from a reconnaissance mission (that the MARV was *returning* can be inferred from it's "move towards" action, which has the coordinates of the MARV's loiter point at (0,0,1)).

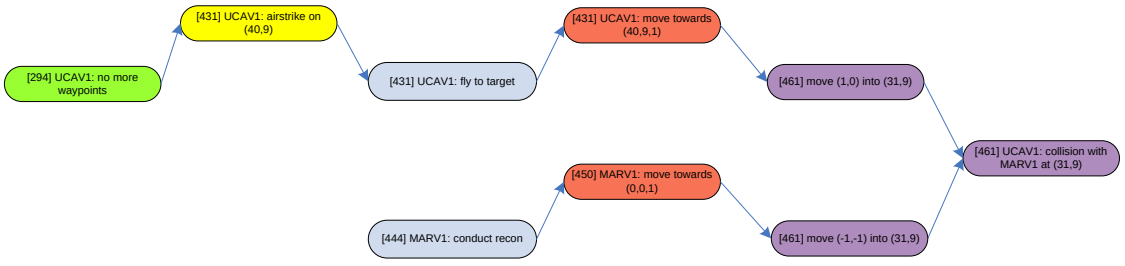

Figure 6.6: Trace for Accident cM1UC1

It is clear from examining the visualisation of the simulation run that the collisions occur because the paths of the two agents cross. Collision avoidance fails because the MARV, which moves second, does not have any collision avoidance behaviour defined.

The relationship of the deviations in the rule to this particular accident is not clear. It can be assumed that the potential for this collision is extant in the behaviour of this system in the absence of any deviations, and that all these deviations have done is to perturb the movement patterns in the air during some runs such that the accident is exposed.

It follows that, if this hazard is valid, then the system is not safe.

6.3.5 Step 5 — Validate Identified Hazards

For the purpose of this validation, we can identify four classes of hazards found in the model:

- Hazards that are clearly artefacts of the model, and are unlikely to manifest in the real system
- Hazards that would very likely be found by a simple manual inspection of the model (without simulation or learning tools)
- Hazards that *might* be found in a manual inspection
- Hazards that would be unlikely to be found without using the simulation-based techniques described here (or another comparable automated method)

6.3.5.1 UC1I2

This hazard is more complex than the others, and hinges on a number of behaviours. Nothing it depends on is *definitely* an artefact. The domain model does not give an explicit mechanism for ensuring that the position of all friendly units is held in the COP (and, in particular, that this functions even for units that are out of radio contact).

Nor does the domain model give any indication that the UCAV uses the COP to avoid friendly fire accidents, or to abort an attack if the target has apparently been destroyed in the interim. For a manned system, assumptions that these will occur might be reasonable (at least during the early stages of SoS development), but the UCAV is an autonomous machine and therefore such assumptions are unreliable.

It is *possible* that an analyst would find this hazard through manual inspection, principally by noting that there were no safety features to protect against such accidents. However, an analyst might also easily assume that the mere presence of a COP would be sufficient to prevent such friendly-fire accidents, along with the confusion over entity identity that was a factor in this accident. In reality, the precise development of the state of the COP over time is important, and the simulation allows this to be discovered.

The hazard might also be difficult to find because there is no obvious relationship between the deviation and the accident. The simulation has made it possible to make this connection by combining the deviation with the configuration of the vignette and the normal behaviour of the system. In effect, the deviation has provided a perturbation of the normal behaviour of the system which has revealed an accident. It is this kind of hazard that it is particularly hard for manual analysis to reveal.

6.3.5.2 TI1

If an infantry unit does not wait for reconnaissance to complete after it detects an unknown entity moving nearby, it may well encounter a threat that it can't deal with. The accident described for this hazard is therefore realistic — it is not just a model artefact. Of course, in reality the infantry could be considered to have graded awareness that improved with as

distance decreased. An entity that was just 'movement' at a distance of several miles might be easily identifiable once nearby.

This hazard could be detected manually. The accident is a direct consequence of the deviation applied (the infantry doesn't wait after calling for reconnaissance, and their movement carries them into range of a threat they didn't know about). The 'wait for recon' stage (and the associated call for air support) is included (as a safety feature) in direct recognition of this hazard. The deviation of this behaviour, and the subsequent accident as revealed by the simulation, have confirmed the 'real-world' necessity of this stage.

Although this accident could have been predicted, it offers an insight into how simulation can begin to be used to explore deviations of operational doctrine and tactics.

6.3.5.3 cM1UC1

Two aircraft sharing the same airspace could be predicted to collide if their tasks lead their paths to cross. However, the explanation of the accident shows that it hinges on the complete absence of collision avoidance on the part of the MARV. This highlights that UAV should not be operating in shared airspace without at least a minimum of deconfliction capability. As with TI1, the simulation is potentially confirming the value of pre-existing safety mechanisms (or, at the very least, prompting a check that they exist in practice.)

6.3.6 Step 6 — Generalise the Results Across Vignettes

In this case study, only one vignette has been implemented and analysed. The hazards cannot, therefore, actually be generalised, but they can be rewritten with a description of the context that they need in order to manifest. The resulting hazard descriptions are given in Table 6.9.

Accident	Generalised Hazard
UC1I2	Multiple infantry engage multiple armoured targets simultaneously
cM1UC1	A MARV and a UCAV operate in shared airspace
TI1	Infantry is operating in the vicinity of an enemy tank, and does not hold its position when it detects an unidentified entity nearby

Table 6.9: Generalised Hazards for Adcock Case Study

Note how the MARV-UCAV collision and the UCAV attack on the infantry do not depend on the presence of any deviations; although they only occur in the studied vignette under deviated conditions, they can potentially occur when the system is functioning normally.

6.3.7 Step 7 — Feedback to Modelling Process

On the basis of the analysis performed during this iteration of the process, feedback can now be provided to the modeller in preparation for future iteration.

It was noted above that the accident cM1UC1 did not appear to be realistic. However, the analyst would suggest that the modeller verify this, and update the model to take at least some account of the results. In particular, the modeller needs to explore the assumptions that were made by the analyst in dismissing the hazard — for example, the assumption that a UAV operating in shared airspace would have some reasonable level of collision avoidance. The technology actually in use should be further investigated, an some suitable representation of its behaviour should be implemented in the model.

A related issue is the 'size of the sky' and how this relates to actual expected movement patterns. It can be observed that the space model used in this model was fairly coarse-grained, and the vignette placed the agents in the SoS fairly close together. It is possible that path crossing would be unlikely in the real system. The analyst should respond to this apparent hazard by seeking assurance that adequate separation would be maintained in reality.

For accident TI1, the relative timings and speeds of infantry movement and organic sensing capabilities of the infantry need to be investigated. The procedures that would be followed by the infantry should also be investigated, along with any information available of the actual behaviour in practice of infantry units in the field (such as studies of how closely they followed the aforementioned procedures).

For accident UC1I2, the modeller needs to look at whether the UCAV should use the COP to avoid carrying out air strikes that are likely to result in friendly casualties. Similarly, they should investigate what procedures (if any) are in place to ensure that friendly units are fully represented in the COP.

6.3.8 Step 8 — Communicate Results

To describe the hazard identified for accident UC1I2 (the most complex and plausible of those discussed in the preceding sections) a fault tree can be constructed. The first version of the fault tree, based only on the highest-probability rule given in Table 6.8, is shown in Figure 6.7.

The fault tree is accurate, in that it correctly depicts that the system isn't safe in the presence of such a deviation, but it's too simple to be useful. Identification of intermediate events is needed in order to show *how* the deviation leads to the accident, thereby providing guidance towards possible solutions or mitigations.

An improved version of the fault tree, incorporating such events, is shown in Figure 6.8. It can be seen that the tree shows how the accident evolves *in terms of the internal mechanisms of the system*. We can now see clearly where safety features are lacking in the system — for example, if Infantry1 has called an air strike while Infantry2 is waiting at the last known location of the target enemy unit, then nothing else needs to go wrong for the accident to occur. Safety features

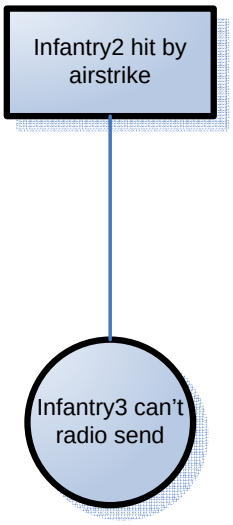

Figure 6.7: Original Version of Fault Tree

could be introduced here, for example by introducing a policy that required the UCAV to check the location of all friendlies before performing an air strike.

6.4 Phase 3 — Evaluate Adcock System

This phase is largely outside the scope of this thesis. However, it is instructive to consider the implications of the analysis for evaluation of the system itself.

As it is, the system does not appear to be safe. Principally, it is relatively easy for the various UAVs involved to collide, for the infantry to move dangerously close to enemy armour, and for the UCAV to deliver air strikes on friendly units.

The safety requirements that are revealed may have validity beyond (reasonable) discrepancies between the model and the actual system. They will apply to a large range of plausible (real-world) implementations of the domain model, and so will be robust to many possible errors in the simulation model. The safety requirements revealed are:

- Adequate deconfliction/collision avoidance features must be provided for all UAVs
- The speed of airborne reconnaissance response needs to be matched to the infantry sensing range and movement speeds
- The COP should contain the location of all friendly units (within some reasonable tolerance for accuracy and timeliness) at all times
- The UCAV should not deliver air strikes near to where infantry are located

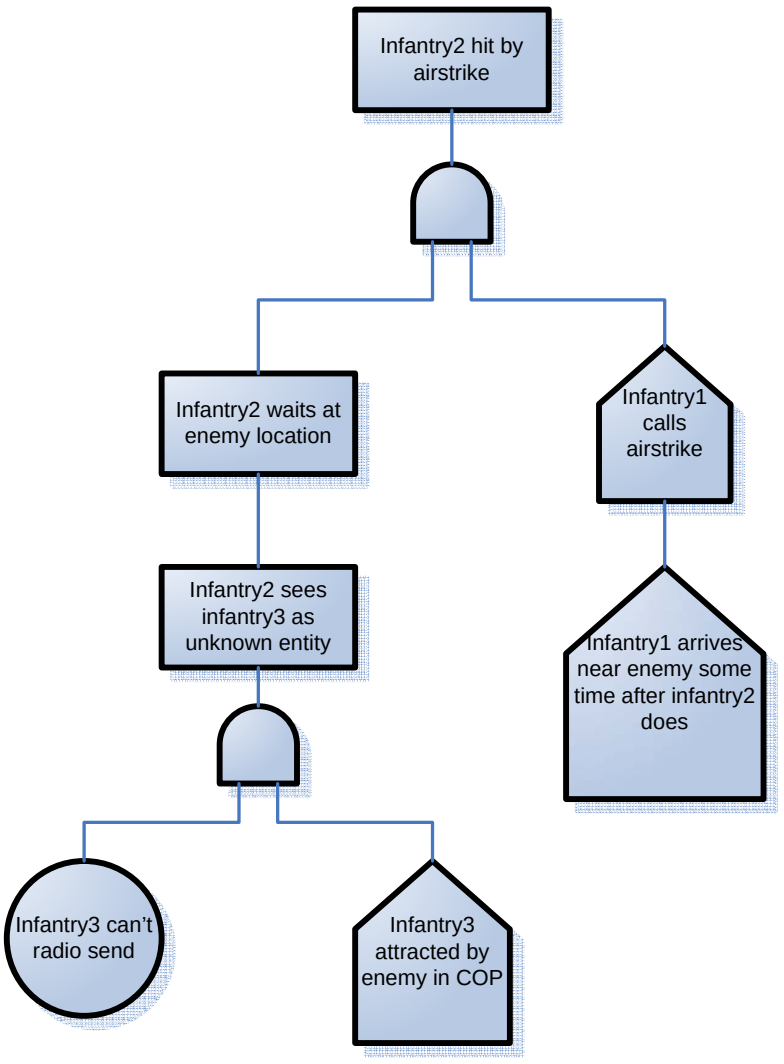

Figure 6.8: Revised Version of Fault Tree

6.5 Phase 4 — Modify Adcock System

Again, this phase is largely out of scope for this thesis. However, as an example, it can be noted that the current SoS as modelled does not have adequate rules for safely coordinating UCAV air support for multiple clients. If this was felt to be representative of the real SoS as defined by the domain model, it might be decided to change the way that the system was expected to operate by defining such rules.

This might proceed by changing the UCAV's behaviour such that it would not deliver an air strike at a location near where a friendly unit is shown in the COP. However, as demonstrated by accident UC1I2, the current setup does not consistently update friendly infantry in the COP. This would need to be supported by a better mechanism for tracking friendly units — all component systems could be required to 'call in' their own position periodically (the specific period

used would be a matter for investigation, and might well depend on the movement speeds of the various entities).

The above mechanism is not sufficient to compensate for the effects of coordinate system errors, because when these are present the 'known' location of friendly units may still be wrong. A mechanism could be added whereby a discrepancy between two position reports in a short space of time would be resolved by the Command agent marking the component system responsible for the discrepancy as not trustworthy for making contributions to the COP.

After phase 4 was completed, the various stakeholders involved would then proceed with another iteration.

6.6 Combined Case Study

The automated nature of the method opens up the potential for an interesting additional case study, which involves taking the vignettes defined for the two SoS models discussed in this thesis (the AGO model that has been used as a running example and the Adcock system that was discussed in this chapter) and running them simultaneously within the same simulated 'world' to see how they interact. Such analysis would be *possible* with a manual technique, but the increase in complexity (especially if multiple pairs of SoSs and vignettes were to be considered) would quickly render it impractical. This is an application of the method where the use of simulation and partly-automated analysis is very valuable.

6.7 Phase 1 — Modelling of Combined System

Because the two case studies used the same implementation environment they can be combined simply by instantiating all their entities simultaneously in the same instance of the simulation engine. Hence no significant modelling effort was required, just a few small changes (for example, to rename the enemies from the Adcock scenario to be Enemy7, 8 and 9 rather than 1, 2 and 3, thereby avoiding name collisions with the enemies from the AGO scenario).

In merging these two case studies, we have *not* merged their mechanisms for maintaining situational awareness by means of a COP. This creates a situation analogous to SoS from two nations operating in parallel against a common enemy — 'blue force' is in fact two forces, each with their own network technology. Component systems of one force are not aware, therefore, of what the component systems of the other force can see. Nor do the two forces explicitly coordinate their actions. (It can be observed that the combined SoS, therefore, exhibits significant *heterogeneity*, noted as a characteristic of SoS in Section 3.1.1.)

It can be noted that the two vignettes, and hence the two SoS models, were easy to combine because they had been implemented for the same simulation framework, using the same space and time representations, and had a variety of common concepts such as definition of enemy types. More diverse concepts, implementations or vignettes would have required additional

modelling work to produce the combined model.

It can be noted, however, that there is considerable interest in the use of the HLA (High Level Architecture) framework for simulation interoperability, as described by Dahmann et al. in [160]; the US Department of Defense, for example, has mandated that all future simulations it commissions be HLA-compliant. The HLA provides a mechanism for diverse simulations to run simultaneously within a single simulated environment, and it provides mechanisms to resolve a variety of interoperability issues, such as time synchronisation. Widespread HLA compliance would increase the practicality of combining diverse simulation models.

It can be observed that although this case study only combines *two* SoS, this is not a limitation of the approach. Given suitable individual models, three or more vignettes could be run simultaneously. It is clear that this would be prohibitively labour-intensive to perform with manual hazard analysis techniques.

6.8 Phase 2 — Analyse Combined System

6.8.1 Step 1 — Acquire Model

As the modelling process described in this thesis has been followed for each model, and changes made to support combined operation as described above, this step is not necessary.

6.8.2 Step 2 — Explore the Space

The deviations defined for the component systems in the two vignettes were combined and runs performed for their powerset as before, using the same probability for each deviation and the same worst probability per run. The same probability (10^{-11} per run) was also used.

6.8.3 Step 3 — Learn Hazard Rules

Initially, running the combined model and learning hazard rules from it had fairly unsatisfying results; several rules appeared that described deviation combinations (of very high probability) that would cause either (a) loss of MARV due to a collision (which would terminate the Adcock SoS mission) or (b) loss of all helicopters to enemy fire (which would terminate the mission of the AGO SoS). Indeed, of the 57226 runs performed, the mission did not complete in any of them. The combined mission explicitly failed in 35407 runs (because either one of the termination conditions was triggered), and timed out in the remaining 21819 (because the combined SoS was not capable of completing both missions). Examples of such rules are shown in table 6.10.

In order to produce some more interesting and plausible rules, the MARV was modified to include some minimal collision-avoidance ability, and the helicopters were given a behaviour that caused them to avoid moving near locations where there was any suggestion at all of a

Accident	Cases	Rule	Probability
eh1	38073	NOT (any deviation on a gun, UAV1 coordinate system skewed, or UAV1 communication failure)	1
cu1m1	5987	NOT (NT unit completely paralysed e.g. by loss of command entity communictions)	1

Table 6.10: Top Hazard Rules for Combined Case Study — Original Version

strong (unsuppressed) enemy presence. Effectively, this provided an additional iteration of the model, which would normally have been performed in phase 4. Were domain experts available, they would have been consulted as to the veracity of the change (for example, would it have been practical to introduce such a rule in practice, and would its effect on other properties of the system have been acceptable?). This might have lead to additional investigations being tasked, which are outside the scope of the current work.

Repeating the same set of runs now produced a number of more plausible, lower-probability hazard rules. The most interesting examples of such are shown in Table 6.11.

Accident	Cases	Rule	Probability
gi3	290	noairsensors_uav4	1×10^{-3}
		!noairsensors_uav4, deviation_inf_find_target_dont_wait_inf1	1×10^{-3}
		!noairsensors_uav4, !deviation_inf_find_target_dont_wait_inf1, noairsensors_uav3	1×10^{-3}
		!noairsensors_uav4, !deviation_inf_find_target_dont_wait_inf1, !noairsensors_uav3, fusiongridnorthwestskew_uav4, !lossofcommsfailure_uav4, !fusiongridnorthwestskew_uav1, !lossofcommsfailure_uav2	1×10^{-3}

Table 6.11: Top Hazard Rules for Revised Version of Combined Case Study

The new rules included 'GI3' — an artillery piece (from the AGO system) hits an infantry unit (from the Adcock system). This is a cross-vignette, cross-SoS accident, and hence is interesting. It will be developed in the following sections.

6.8.4 Step 4 — Explain Hazard Rules

The top rule for GI3 in Table 6.11 consists of the single term 'noairsensors_uav4'. However, all the four rules shown are of equivalent probability.

On investigation of the runs corresponding to the rules, it emerges that there are few examples that actually correspond to accidents occurring when (noairsensors_uav4) is a deviation. Looking at the decision tree that is learned, the first rule in the table covers only 9 of the 290 accident cases, while the fourth rule (fusiongridnorthwestskew_uav4) covers 126 of them (along with 1

non-accident case; a false positive). The fourth rule will therefore be considered here.

To apply the tracing tool to the combined case study, the domain-specific tracing rules derived for both the AGO and Adcock scenarios can be combined. The application of the tracing tool to explain the accident can then be performed, and can be seen in Figure 6.9.

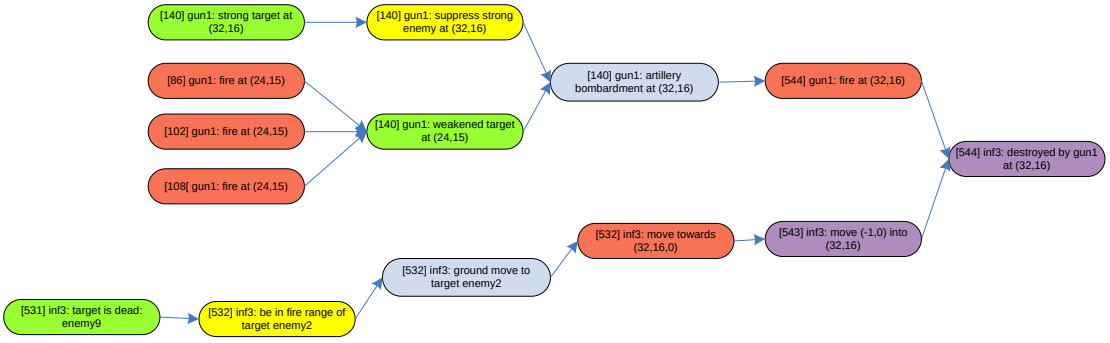

Figure 6.9: Trace for Accident G1I3 in Combined Vignette

In the case of this rule, it can be seen that the skew applied to UAV4 means that enemy1 is no longer on any patrol route, and is therefore able to attack and destroy the helicopters as they move past on their way to other targets. Consequently, the helicopters are all destroyed. This means that the AGO system no longer functions, and therefore it does not destroy any of the enemy that it normally would.

The task of destroying the enemy therefore falls to the Adcock system. Infantry3 therefore sees enemy2 in the COP and moves to attack it, but it arrives during an artillery barrage by gun1 and is destroyed.

6.8.5 Step 5 — Validate Identified Hazards

The explanation found in Step 4 appears plausible. However, while uncovering that explanation it can be observed that there is an oddity of the model that draws its validity into question. Some time before Infantry3 is due to arrive and hit by the gun that's targeting enemy2, Infantry2 arrives in the adjacent grid square (where enemy6 is located). It should destroy enemy6, and (given that infantry can fire into adjacent grid squares) destroy enemy2 without moving into the artillery fire zone (which covers only a single square). However, this does not occur. Inspection of the two models reveals that, due to incompatibilities between the way combat is resolved in the two models, the Adcock infantry cannot damage the AGO enemies.

Correcting this error causes the accident no longer to manifest in the simulation. Infantry2 destroys enemy2 and enemy6 without exposing itself to friendly artillery fire. It is therefore potentially possible to dismiss this rule as being merely an artefact of the simulation. However, it can be noted that it is still possible, in the absence of coordination between the AGO artillery and the Adcock infantry, that the infantry could enter a grid square that was being bombarded

by the artillery. This is more likely than might be expected because the infantry will be drawn to move towards the same targets (enemy units) that the artillery will be drawn to fire at. Even allowing for the ability of the infantry to attack a target from an adjacent square, any inaccuracy on the part of the artillery could potentially make them vulnerable.

It is reasonable, therefore, to conclude that there is a plausible hazard here. In the current context, even given the problems with the model identified above, it can reasonably be observed that the system is not safe.

6.8.6 Step 6 — Generalise the Results Across Vignettes

As with the two vignettes considered individually, since the combined vignette is only a single vignette for the combined system, there are no other vignettes to merge it with. However, we can observe that a friendly-fire hazard is present whenever the Adcock infantry are operating in the same area as the AGO system and enemy units. This situation is sufficient for an accident to occur without any abnormal behaviour.

6.8.7 Step 7 — Feedback to Modelling Process

The most important concern for the next iteration of the model is to verify what ability the MARV actually has to avoid collisions, and whether helicopter crews are likely to use the shared picture to avoid exposure to suspected enemy sites. These issues, however, were raised in the individual case studies (see Sections 6.3.7 and 5.10.1).

New questions arising out of the combined vignette relate to the ability of the Adcock infantry to destroy the AGO enemy units, and to the coordination measures (if any) that would be imposed between the two military units if they were deployed in the same theatre. The analyst would want assurances (and evidence) that adequate coordination was in place. Lack of coordination between two parts of a force has been implicated as a cause in many friendly fire accidents (see Regan in [15] for several examples), so a potential coordination-related friendly fire hazard is a cause for serious concern.

6.8.8 Step 8 — Communicate Results

As shown previously for the AGO and Adcock case studies, in this step the analyst would use a Fault Tree or similar notation to express the hazard in a form that would be generally intelligible to safety engineers. This will not repeated here.

6.9 Phase 3 — Evaluate Combined System

As it is, the combined operation of the two systems does not appear to be safe. Even when additional safety measures are imposed (helicopter avoidance of suspected threats, and MARV

avoidance of other aircraft) there is still an unreasonably high probability of an Adcock infantry unit being destroyed by one of the guns from the AGO SoS. This, of course, relies on the precise location of the enemy and the timing of the artillery barrages and infantry movement, and will therefore need further investigation. It is clear, however, that any combined used of the Adcock and AGO systems will need to coordinate artillery fire and infantry movements.

6.10 Phase 4 — Modify Combined System

As noted in the Adcock example, this phase is largely out of scope for this thesis. However, it can be observed that the immediate priority arising from the combined scenario investigation is the need for explicit coordination between the two SoS. This could potentially be achieved by requiring that the AGO artillery not fire on locations where *any* friendly units were located. For this to be effective, however, there would need to be a mechanism for detecting and communicating with arbitrary allied entities so as to acquire the necessary information.

Once any additional safety measures for combined operation had been defined, and implemented in the simulation, the analysis could be repeated. Although absence of the accident in the combined system would not be sufficient evidence to consider this hazard resolved, *presence* of the accident would be a strong indication that the changes were inadequate. Repeating the analysis might also reveal any *new* hazards that the changes had introduced. The use of simulation, machine learning and agent behaviour tracing would allow the analysis to be repeated with only a modest effort, in contrast to the major effort involved in repeating an entire systematic manual analysis. This practicality of iterative repetition is a major benefit of the hazard analysis approach presented in this thesis.

6.11 Summary

This chapter introduced the Adcock SoS case study and walked through the hazard analysis process presented in Chapters 3 through 5. This revealed a number of interesting hazards, one of which would have been difficult to discover by hand in a complex system. It was then demonstrated that the implemented versions of the two SoS models could be run simultaneously in the same environment, and a hazard that emerged from the interaction of the two SoS was discussed.

One of the most significant things that this case study demonstrates is the ease with which we can combine multiple vignettes and SoS and explore the interactions between vignettes through a common world model. Such an activity falls well outside the capabilities of manual hazard analysis techniques.

The next chapter evaluates the work presented in Chapters 3 through 6 against the requirements identified in Chapters 3, 4 and 5, and against the thesis proposition that was presented in Chapter 1.

Chapter 7

Evaluation

Chapter 1 presented the thesis proposition, which stated the central claim of this thesis. This chapter evaluates that claim, looking at the extent to which it is supported by the evidence presented in this thesis. This task will be performed in two complementary ways:

First, Chapters 3 through 5 have introduced a large number of explicit requirements that were derived from the thesis proposition and from existing work in the literature. In order to support the claims of this thesis, the approach must at least minimally meet these requirements. Sections 7.1 through 7.3 provide an evaluation presented in these terms.

Second, in Section 7.4 the work is evaluated against the thesis proposition presented in Section 1.3, drawing on the results of the case study presented in Chapter 6.

7.1 Evaluation Against Overall Process Requirements

In Chapter 3 a set of requirements were identified for the overall SoS hazard analysis process. These requirements are concerned with the practicality and utility of any SoS hazard analysis approach. If these requirements are satisfied, the method described in this thesis will be suitable for incorporation into a wider safety engineering process.

Requirement P1 — Process must be applicable early in lifecycle The approach described in this thesis uses MODAF models as its (initial) source documents, and as noted in Section 4.4 these are likely to be available early in the lifecycle. Such documents may, especially at first, contain errors and omissions, but the iterative nature of the process and the interleaved cross-check and validation steps give confidence that such problems will be addressed. The use of multi-agent simulation means that the system can be modelled at the level of detailed required (or possible) given the detail of the system descriptions available.

Requirement P2 — Process must tie in with other safety activities A SoS safety lifecycle was described in Section 3.2. This was derived by combining a widely-accepted model of the

conventional safety lifecycle with a recently-proposed model of the general SoS engineering lifecycle. Section 3.7.2 built on this to discuss the role of the thesis approach within the lifecycle. The work in this thesis is most relevant to the hazard identification and risk assessment activities conducted during the architectural design phase, but value for later stages was also identified.

To allow the results of the analysis to be integrated into a broader safety process, and to be used by safety engineers not familiar with this method, Section 5.11 showed how the results could be expressed as fault trees. The results of the decision tree learning provide a starting point, which would then developed by the analyst with help from the tracing tool.

Requirement P3 — Process must support iterative modelling and analysis As noted in Section 3.6, the use of multi-agent simulation means that the behaviour of the simulated SoS emerges from the actions and interactions of the agents that compose it. If a change needs to be made to the behaviour of an agent, the agent can be modified and the new emergent behaviour observed.

In practice, changes to one part of the model (e.g. to one agent) will require changes to other parts. For example, if one type of agent changes the protocol it uses for distributing sensor data, the agent types that are its peers may have to change as well.

The approach described in this thesis guides the modeller towards producing constructive simulations based on high-level architectural models. Such simulations require a relatively low level of computational power compared to, for example, high-fidelity flight simulators, so large numbers of runs can be performed quickly using modest hardware. As noted in Section 5.5, the execution of the simulation runs is easy to parallelise.

This thesis has assumed deterministic models (and such have been used in the case studies). The use of probabilistic models is desirable. However, such models will require a substantial increase in the number of runs performed, and therefore in the volume of data that must be processing by the machine learning algorithms. For these reasons, the use of probabilistic models lies outside the scope of this thesis. Section 8.2.2 raises this as possible future work.

The analysis approach described remains only *semi*-automated; manual input is required to all steps of the analysis process. The learning of hazard rules will, however, quickly show if previous hazards are exhibited in the output of the revised model. This provides an automated check that changes to the model have had the desired effect (of eliminating hazards).

Of course, the effort required to repeat an analysis will depend on the scale of the system being analysed. Effective use of this method will require engineers and managers to allow sufficient time and personnel for repetition of analyses.

Requirement P4 — Process must be feasible and practical to perform For the SoS such as this thesis describes, development times are likely to be long and their budgets are large. It is reasonable to suggest that a team tasked with hazard and safety analysis of a SoS will be

able to command substantial resources. Small examples have been performed in the space of a PhD research programme without using external resources for the actual development of the case study models or performing their analysis, so it is likely that scaled-up uses will still be affordable.

It was also noted in Section 3.6.1 that the fidelity and detail of the model, and the depth to which the analysis is performed (e.g. of combinations of deviations) can be adjusted in order to control the modelling effort and computation required. The effort required can therefore be managed.

When analysis is carried out over a long period of time, the model can be maintained throughout that period and therefore the initial development costs are less significant. In practice, it seems likely that this will be the case, with hazard analysis models being maintained for the whole lifetime of the SoS. This is supported by the move, already noted above, towards simulation-based development and acquisition practices.

As noted in Section 3.6.1, constructive multi-agent models are already widely used in the military and aerospace domain, and there is strong interest in increasing their use (see, for example, Oren et al. in [161] for a discussion of multi-agent techniques in simulation-based acquisition). It is therefore possible, for any given project, that such models will already be available, or that there will be stakeholders that are interested in constructing a model for purposes other than safety analysis. Such models may be easily adapted for the method described in this thesis.

Requirement P5 — Process must provide visibility to engineers The use of multi-agent models means that the behavioural rules of each individual agent can be reviewed and comprehended individually, and offers the potential for the emergent behaviour observed in simulation to be traced back to individual agent behaviours and properties.

The provision of structured logging output and potential for visualisation makes it practical to discover how the conclusions of each run were reached. The machine learning and agent tracing tools that are defined in Chapter 5 likewise relate the observed overall behaviour back to the original models.

A question can be raised as to the comprehensibility of agent models, even after substantial analysis. The alternative is high-level models in which the basic element of modelling is not the discrete 'object' (such as an agent) but the *process*. Mao et al., in [162] notes that multi-agent simulations can be difficult to understand, and proposes systems dynamics models as an alternative (for the task, in their case, of modelling virtual communities). However, Parunak, Savit and Riolo, in [163], note that many complex systems (especially systems with even moderately complex components) are difficult to model accurately using such approaches.

It can also be observed that although the domain model (expressed in augmented MODAF or a similar notation) and the design model (expressed here in the Prometheus process artefacts) may be easily comprehensible, the resulting implemented models (expressed, perhaps, in a general-purpose programming language) may not be. Indeed, this was part of Lam's original

motivation for his work on agent tracing [154].

Section 4.6 proposed that a log be kept of all additional information acquired (or assumed) during the development of a model. Combined with the source model, this provides an additional measure of visibility into the 'roots' of the model, i.e. into the information from which the model was derived.

Questions can be raised, of course, as to how scrupulously this rule will be observed in practice, as it generates additional work that in many cases will not be needed. In addition, many acquisitions of information are not formally acknowledged, and therefore are unlikely to be recorded. Assumptions, in particular, are often not conscious.

Similarly, the modeller and analyst will inevitably start with considerable tacit knowledge, and their modelling decisions will be made in the light of this knowledge. Such knowledge may not be recorded.

7.2 Evaluation Against Modelling Requirements

In Chapter 4 a set of requirements were identified for the development of SoS simulation models. These requirements are concerned with the fidelity, flexibility and ease of analysis of the models used as the basis of any SoS hazard analysis approach.

Requirement M1 — Models must adequately represent SoS The nature of multi-agent simulation is such that in order for a model to produce the same result as the real system (in the normal case), the model must have an internal behaviour that is equivalent, at some level, to the internal behaviour of the real system. The overall behaviour then emerges from the internal design.

It is not possible to guarantee, however, that a model that produces expected behaviour under normal circumstances will produce accurate (or even sensible) predictions of behaviour in deviated cases. Inevitably, the models will be much simpler than the real systems, dramatically so in the case of SoS because SoS contain humans with a wide range of responsibilities and behaviours.

This is the general weakness of the 'positivistic economics' approach to model validation (the claim that a model of a system is adequate if it produces the same observed behaviour as the real system; see Sargent in [88]) — predictions outside the range for which real-system data is available remain highly suspect. See Richardson in [126] for some commentary on the problem of extrapolating beyond the data in a nonlinear system.

In this thesis, this is addressed by the use of iteration, with validation achieved through consultations with domain experts, the use of more detailed (partial) models, and improvements to the model being made for future iterations. Guidance to help address these issues is provided in Chapters 4 and 5.

It was shown in Section 4.8.1 that the Prometheus process and its associated notation supports the *implicit* representation of all critical SoS characteristics. The specific representations of those characteristics were identified (for example, the characteristic 'communications' is represented by protocol and message descriptors). It follows that, with an appropriate operational model and an accurate implementation, these characteristics will be carried through into the running simulation.

It can be noted, however, that the Prometheus notation and process don't provide *explicit* support, or guidance, for all characteristics. For example, although component goals can be captured implicitly in plans, there is no way to explicitly represent them in the notation, and the Prometheus process does not guide the modeller to elicit and record them. A skilled modeller, however, can address these deficiencies whilst still benefitting from the guidance provided by the process.

Requirement M2 — Models must capture realistic SoS contexts The runs that are executed in simulation are based on deviations of vignettes which are derived from the scenarios, and the scenarios are derived from the domain model (which should capture the expertise of domain experts) and further consultation with domain experts. They can therefore represent the best understanding of the actual use of the SoS that exists at the time that the model is developed. This includes the details of the environment that the simulated SoS encounters.

The process does not provide a method or guidance for deriving environments that are of particular concern for the system under analysis. The choice of environments for this purpose must be informed by past experience and domain expertise.

Requirement M3 — Models must incorporate plausible deviations The modelling process described in Section 4.3 leads the analyst to derive deviations of normal agent behaviour and the environmental conditions expected in the vignettes.

An attempt to provide coverage of agent deviations is provided by combining a broadly acceptable model of the structure of an agent with a set of well-established guidewords for deriving deviations. Given that these are, ultimately, only stimuli to *increase* the range of deviations that engineers will propose, it is reasonable to suggest that this provides adequate coverage of the deviation space.

It can be noted, however, that there is no established theory of how to achieve adequate coverage of deviant behaviour for entities as complex as the ones that are being modelled here. Similarly, although the modeller is encouraged to explore a variety of different vignettes, it remains difficult to guarantee the level of coverage of the possible SoS-level behaviours achieved by any given set of vignettes.

Requirement M4 — Models must provide the features needed for analysis The granularity of the model is derived from the granularity of the MODAF model (thereby capturing the

expertise of domain experts), and is reviewed during the modelling process. There is therefore a measure of confidence possible in the granularity and detail chosen for the model. In practice, it is likely that iteration will be required to identify the division of the SoS into model agents, and the level of detail of those individual agents, that is optimal for revealing all hazards.

High-level concerns are identified early on in the modelling process (step 2), and they are traced through all the distinct stages of the model's development. Cross-checks at various points help to prevent 'drift' as the model is developed, helping to ensure that actions taken to embed concerns in the model at one stage are preserved through to later stages.

Requirement M5 — Models must be verified and validated Validation of the developing model is specified at various points in the process, such as the validation of the design model required in step 5 (see Section 4.8.4). Furthermore, the cross-checks and the tracing of concerns help to ensure that the domain knowledge embedded in the domain model is preserved accurately into the implementation.

Consistent with Drogoul's description of multi-agent modelling (see Section 4.1.2), the process is explicitly broken down into stages so that the roles of different skill sets (domain expert, modeller, implementor/software engineer) are clearly demarcated and the need for communication between them is clear.

The fidelity of a model can be improved through multiple iterations, as it exposed to further validation. In particular, attempts to validate specific results of the model (as described in Section 5.8) will cast light on its predictive accuracy.

Validation remains, however, a general problem with simulation modelling of all kinds. Dunnigan, in [164], notes that many of the military performance models produced by the Operations Research community in the 1960s and 1970s failed utterly to predict known historical results when the comparison was finally made. Sargent, in [88], discusses a variety of ways to validate simulation models, but few are directly applicable in the current context.

Requirement M6 — Models must be amenable to analysis The modelling process described in this thesis leads to models that have a wide range of deviations specified for them, so that they are suitable for the derivation of hazard rules using a decision tree learning algorithm. The process of tracing concerns helps to ensure that all accidents occurring in simulation are detected and reported, allowing rules to be learned for them.

Additionally, agents are described (from the design model onwards) in terms of Belief-Desire-Intention concepts. This renders the logs of their behaviour intelligible to a human reader and amenable to explanation by an agent tracing tool. It also provides a natural set of internal agent events that can be logged, increasing the likelihood that logs will contain all relevant information.

7.3 Evaluation Against Analysis Requirements

Chapter 5 identified a set of requirements for a hazard analysis method that works over the simulation models of SoS described in Chapter 4. These requirements are concerned with the capability, validity and transparency of the analysis method and results. Satisfaction of these requirements gives confidence that the analyses performed will be accurate and adequately complete, not only with respect to the model but with respect to the real system.

Requirement A1 — Analysis must selectively explore the space provided by deviations The execution of runs corresponding to the powerset of the defined deviations (with depth limited by combinations of the probabilities of those deviations) allows the plausible region of the deviation space to be explored while ignoring the (vast) improbable region and making the deviation space tractable.

Requirement A2 — Analysis must find all paths to accidents The machine learning approach to deriving hazard rules described in Section 5.6 allows multiple rules to be derived for each accident, potentially corresponding to multiple routes. Analysts can use the set of derived rules to guide them to uncover all the pathways that have distinct corresponding rules. Whether all possible rules are extracted from data will depend on the efficacy of the machine learning algorithm used. For example, in the case of Quinlan's C4.5 decision tree learner adopted in this thesis, the decision tree generated is not guaranteed to be a perfect classifier in cases where the parameters of the training instances do not lead to unique outcomes.

It can be noted that, potentially, one rule expressed in terms of deviations could cover several distinct paths, in terms of the internal model mechanisms they exploit.

Requirement A3 — Analysis must lead to results being validated Section 5.8 proposed a method for validating the hazards that are found by the simulation analysis. In practice, such validation is inevitable because the analyst will have to motivate the existence of the hazard to other engineers and stakeholders, and unrealistic hazards are likely to be rejected at this stage.

Requirement A4 — Analysis must present results in human-comprehensible form The hazard rules derived by machine learning are expressed as human-readable proposition logic expressions. Similarly, the explanations provided by agent tracing tools are human-readable tree structures. As noted by McIlroy and Heinze in [95], the use of mentalistic 'folk psychology' concepts in Belief-Desire-Intention representations of agent behaviour make them readily comprehensible to non-specialists.

Furthermore, Section 5.11 provides guidance on representing the results of analysis in traditional safety artefacts such as fault trees. This is similar to their well-established use in accident analysis [9].

It was noted in Section 5.6.3.6 that there is a tendency for the learned decision trees to be cluttered by large numbers of negative terms ('NOT deviation1 AND NOT deviation2', etc). It was suggested that such terms represented 'normal' states or events, and that they could therefore be treated as occurring with probability 1 (thereby removing them from the rules derived from the tree). This is well established in the case of fault trees (where normal events are often assumed to occur with probability 1), but doing this automatically risks hiding information; it is possible that such a negative term represents a deviation that can *prevent* an accident (which was caused by other deviations).

To clarify: it is not the case that a 'NOT deviation X' term in a rule substantially affects the significance of the rule for the safety of the system (because the probability of the deviation being present, and thereby preventing the accident, will be low). This is commonly known as the Law of the Excluded Miracle — it is not reasonable to ignore an accident rule because it could *possibly* be prevented by some specified state or event. Instead, a probability must be assigned to the 'miracle' and the rule reconsidered on that basis. With this in mind, it is reasonable to exclude negative terms because they have little effect on the probability of the rule.

If an analyst knows that a negative term can prevent an accident occurring then they can use this information to guide the development of the system. For example, if the 'deviation' could be made part of the standard operating state of the system (greatly increasing its probability of being present in any situation) then it would be possible to use this to increase safety. There is a risk that if negative terms are discarded then such information may be lost.

It may be that the condition 'NOT deviation X', while not 'wrong' at the component system level (in the sense of being unintended, or being in violation of a specification), is still needed for in order for a particular hazard to occur. As observed in Chapter 1, it may be that the composition of two individually acceptable behaviours causes an accident.

There is a further issue related to the combination of rules with negative terms. The *consensus rule* is a well-known result in logic:

$$(A \wedge B) \vee (\neg A \wedge C) \equiv (A \wedge B) \vee (\neg A \wedge C) \vee (B \wedge C) \tag{7.1}$$

If there is a rule for a given accident of the form 'deviation A AND deviation B', and there is another rule for the same accident of the form '(NOT deviation A) AND deviation C', then the policy of removing negative terms will leave 'A AND B' and 'C' as rules. It may well be the case that deviation C is not sufficient to cause an accident, and this will be revealed when the rule is studied in detail. The analyst will probably, therefore, discard rule 'C' as an artifact of the analysis tools. Given the consensus rule, however, it follows that 'B AND C' also needs to be investigated (regardless of the state of A).

One final point can be made: a rule '(NOT deviation A) AND (NOT deviation B)' is *not* necessarily equivalent to 'no deviations' i.e. the normal case. That rule says nothing about

the presence or absence of deviations other than A or B. For example, that rule also covers the cases 'deviation C' and '(NOT deviation C) AND deviation E'.

Requirement A5 — Analysis must present results in terms of SoS mechanisms The use of agent tracing tools helps to find the causal patterns at the level of the mechanisms of the model that produce accidents. Such patterns potentially provide a strong basis for the comprehension of within-model causal relationships. The tool can find causal patterns in logs of great size and complexity, including logs that a human would have great difficulty comprehending manually. The tracing tool operates at the level of individual vignettes, but guidance is given in Section 5.9 to help the analyst generalise results across multiple vignettes.

In the case studies there were a number of cases where an event could not be explained (or could not be *adequately* explained) by the tool, given the rules implemented. This potentially could have been addressed by the formulation of additional tracing rules.

The development of effective tracing rules, and selection of appropriate log output, to maximise the utility of the tracing tool analysis requires human expertise. This is particularly true in the case of explanations that hinge on space or time relationships.

7.4 Evaluation Against Thesis Proposition

The central proposition of this thesis was given in Section 1.3 as:

> It is possible to identify emergent hazards in Systems of Systems, not easily identified using conventional manual hazard analysis, through the adoption of systematic and exploratory multi-agent simulation and analysis using machine learning and agent behaviour tracing techniques.

The work described in the thesis can now be evaluated against that proposition.

7.4.1 Systematic Development of Multi-Agent Simulations

Chapter 4 provides a systematic, repeatable process for deriving an executable simulation model from an architectural model of a SoS. This model incorporates a description of the SoS (derived from a MODAF model) and the various environments that it will encounter (derived from consideration of the probable scenarios and vignettes).

7.4.2 Identification of Emergent Hazards

When the model is executed, the agents it comprises act and interact according to their own behavioural models. The behaviour of the overall SoS emerges from this. This may include accidents, which arise from a combination of normal behaviour and deviations applied.

The resulting model is annotated with a set of deviations, derived using a systematic method that guides the modeller to consider a wide variety of relevant possibilities. Chapter 5 gives a method for exploring the plausible combinations of deviations (while limiting the search by excluding those that are improbable), and identifies tools that can be used to extract hazards from the resulting data, and aid in comprehension of how the system exhibits them.

There is a pervasive difficulty related to evaluating the thoroughness of the method. This has two forms: the extent to which the model expresses the hazards present in the real system, and extent to which those modeled hazards are uncovered by the analysis. It is possible that a hazard can occur in the real system but is not represented in the model, and it is possible that a hazard is present in the model but never revealed during analysis. In both cases, the advice in this thesis can be seen to provide set of heuristics, always with the explicit aim of identifying the maximum number of serious hazards. The extent to which the hazards in the real system are identified will depend on the quality of the heuristics defined.

The heuristics used in the method present a problem for evaluation because it difficult to assess their quality. For analysis heuristics, an analogy can be made with the concept of coverage in software testing. It would be useful to have available some corresponding indication of how thoroughly a simulation has exercised the system in attempting to find an accident scenario. This would enable us to say whether adequate simulation had been performed. Such 'stopping' or 'adequacy' criteria are beyond the current state-of-the-art but could well be required if simulation is to form a convincing component of a safety case.

Some confidence that the heuristics have significant promise can be gained through industrial experience and empirical studies (see Section 8.2.7), but explicit further work on evaluation of heuristics would be valuable.

7.4.3 Applicable to SoS

The Adcock case study described in Chapter 6, and the AGO study that has been used as a running example, demonstrates that the approach described in this thesis is broadly applicable to SoS of significant scale and complexity. The systems used, although relatively simple, are representative of real SoS that are being deployed and used now and in the near-term future.

The Adcock example exhibits all of the SoS characteristics identified in Section 3.1.1. For example, the infantry are geographically distributed across the area, and move autonomously guided by their own local situational awareness. They collaborate with the rest of the system, and this collaboration is coordinated by asynchronous requests for fire and reconnaissance support. In addition, the AGO example covers many of the characteristics. For example, although (as implemented) the situational awareness model is entirely shared between all component systems, the component systems move and act autonomously in response to that central model. Both case studies are, therefore, representative models of SoS.

Several interesting and plausible results have been produced from the vignettes studied, suggesting ways in which the SoS modelled are insufficiently safe. Although the nature of the

analysis performed here is such that it cannot be confirmed that the identified hazards exist in the actual systems, a case has been made that they are worthy of further investigation.

7.4.4 Hazards Could Not Easily Have Been Found by Manual Means

It is *possible* that some of the hazards identified in the case studies could have been discovered by a manual analysis. With the benefit of hindsight this can be difficult to judge. Assessing whether this would have been likely *when analysing an SoS of real-world complexity* requires identification of the *systematic* manual method that would have reliably revealed those hazards. One can then judge whether the level of effort required to do this would be practical for such large SoS.

It is *possible* that the hazard described for the accident UC1I2 (see Section 6.3.4.1) could have been discovered manually. An attack by the UCAV on a location containing friendly infantry was always possible because there are no explicit safety measures to prevent it. However, to reveal that it is actually plausible in the SoS as defined requires consideration of the interaction of the *normal* behaviour of several component systems. In the case study it is relatively easy, because there are few types of component system and few instances of each, but in a larger SoS it would become increasingly impractical.

It is clear that reliably discovering this hazard in analysis requires many aspects of system behaviour to be considered. Finding the hazard would require consideration of the interaction of multiple clients issuing air support requests, a consideration of the effects over time of an infantry performing its attack behaviour and its "wait for reconnaissance on unknown entities" behaviour, and consideration of the awareness that one infantry unit would have of another which was out of radio contact. In all cases, the spatiotemporal aspects of the behaviour would depend very much on the context of a particular vignette. Although an analyst *might* manage to do this, they could not be guided to do so using a systematic method. It therefore cannot be relied on. By contrast, the method described in this thesis can easily address all these aspects simultaneously.

To find the hazard for the combined vignette accident GI3 (from Section 6.8.4), an analyst would need to consider the two vignettes being performed simultaneously. The hazard arises from the dynamic interaction of the behaviour of the SoS component systems in a particular context, in that the movement of the infantry and the targeting of the guns is drawn towards similar locations, with context determining whether they coincide in time. It can also be noted that there is no direct, obvious connection between the accident and the deviations that cause it to occur; a manual brainstorming effort using the same cues could not reasonably be expected to reveal the hazard, which is inherent (although not obvious) in the normal behaviour of the two SoS.

On the basis of the case studies, it is reasonable to claim that the approach described in this thesis can find hazards that would not easily be found by existing manual means.

Chapter 8

Conclusions

This chapter concludes the thesis by summarising the key contributions that have been made in the preceding chapters and identifying opportunities for future work.

8.1 Summary of Thesis Contributions

The thesis has made contributions in three core areas:

- It identifies the key challenges for SoS hazard analysis, and condenses them into a set of explicit requirements that clearly defines what is needed from modelling and analysis techniques to support the identification of SoS hazards.
- It provides a systematic method for building multi-agent simulations that approximate the behaviour of component systems operating as part of a SoS in a variety of operating contexts and realistic scenarios. This method involves taking high-level models of SoS expressed using MODAF and systematically translating them into agent models using an adaptation of the Prometheus method.
- It defines a hazard analysis method based upon machine learning and agent behaviour tracing that extracts and explains the significant contributory causes of accidents identified through simulation.

8.1.1 Requirements for SoS Hazard Analysis

Chapter 2 provided an extensive review of the literature, including work on the properties of SoS, on the difficulties that they present for safety engineering, and on a wide range of existing hazard and safety analysis techniques that might be useful for SoS analysis. A range of problems and deficiencies were identified with these existing techniques.

Chapter 3 extended this by presenting a set of characteristics common to many SoS, and outlined a variety of ways by which each characteristic could lead to hazards. The key terms

"SoS accident" and "SoS hazard" were defined and explained, thereby clarifying the subset of hazards occurring within SoS that are of particular concern in this thesis.

The above allowed a set of requirements for an effective SoS hazard analysis process to be defined. The requirements at the whole-process level were given in Chapter 3. The requirements for the modelling and simulation of SoS were given in Chapter 4, and those for analysis of the resulting model in Chapter 5. These requirements define how practical hazard analysis of SoS is significantly different to the analysis of simpler types of system, and thereby allowed the work presented in this thesis to be developed and evaluated.

8.1.2 Method for Building SoS Simulations

To support development of SoS analysis techniques, a SoS safety lifecycle was defined. This provides the context for the modelling and analysis activities defined in the rest of the thesis. Based on the overall process requirements identified previously, multi-agent simulation techniques were introduced as a means of addressing the problems of SoS analysis. Chapter 3 presented a high-level process for the use of these in hazard analysis, and explained in outline how the properties of the multi-agent models matched the requirements of SoS analysis. Crucially, it was observed that multi-agent simulation can reveal SoS behaviours that emerge from the interaction of the component systems.

Chapter 4 built on this to present a systematic method for turning high-level architectural models (in the MODAF format that is becoming widely used in the UK military) into executable simulations. The process provides guidance for the identification of concerns specific to the SoS under analysis. These form a thread throughout the modelling process that allows the modeller to verify at each step that the developing model is consistent with the needs of the particular analysis.

The modelling process leverages the existing Prometheus multi-agent design method, thereby allowing the guidance and tool support for this method to be exploited. Prometheus supports agent design in terms of the Belief-Desire-Intention model of agency, and this is crucial for allowing effective tool support for analysis. The process leads the modeller to develop a range of appropriate deviations that can be applied to explore the possible variations on expected SoS behaviour, using developments of established agent architectures and hazard analysis guidewords.

8.1.3 Process and Tools for Simulation Hazard Analysis

Chapter 5 presented a process for analysing the simulation model so as to efficiently explore the space of possible behaviours that the model could exhibit. Supporting techniques, in the form of machine learning and agent tracing were applied to assist the analyst in finding the hazards that are present in the simulation model. The tools provide human-comprehensible output that describes not 'what happened' or 'what is probabilistically likely to happen', but rather 'how it

happened' in terms of component system-level events in the model. Like the simulation model itself, they support, rather than replace, the analyst.

Guidance was provided for incorporating the results of the analysis into the SoS safety lifecycle, and for translating it into formats (such as fault trees) that are widely used and understood by safety engineers. The combination of automated analysis with ease of analyst comprehension goes some way to tackling the common question raised as to the admissibility of simulation evidence in the safety engineering process, thereby potentially allowing the techniques described in this thesis to see widespread adoption.

8.2 Future Work

The approach to SoS hazard analysis presented in this thesis is almost entirely novel, and it has therefore raised many interesting areas of future work.

8.2.1 Guidance on the Use of More Sophisticated Models

It can be observed that this thesis provides modelling guidance centred on the use of high-level architectural models, and on the derivation of component system behaviour from explicit sequences and procedures. There is great potential for guidance on the use of more sophisticated space, time and cognition models. Aspects of the latter include general-purpose team coordination, adaptation to detected faults, and realistic representation of human perception or decision-making (see, for example, Norling in [165]). Potentially, such improved models could increase the scope and power of the hazard analysis.

It is important, however, that any increase in modelling effort, validation effort, or computational load is matched by a gain in hazard analysis effectiveness. Further research on ways to increase model detail in ways that genuinely improve analysis would be valuable.

8.2.2 Use of Stochastic Models

The work in this thesis has used wholly deterministic simulation models to aid performance, ease of development, and ease of analysis. Lucas, in [166], argues that although deterministic models are still widely used in military modelling and simulation, stochastic models have much greater expressive power and can lead to more realistic results. It would therefore be valuable to extend the method in this thesis to allow the use of stochastic models. As well as requiring issues such as the number of replications needed of each run to be addressed, this would require significant changes to the way that analysis is performed (principally because a given set of deviations is no longer uniquely associated with a single outcome).

8.2.3 Improved Risk Measures and the Derivation of Risk-Space Models

The work described in this thesis has concentrated on the use of discrete deviations (boolean-valued parameters) to produce discrete accident results. The low probability of accidents in the real world means that the probability of accidents in a realistic, moderate-fidelity simulation is likely to be low — a small space or time difference can easily turn an accident into a near-miss. Crude deviations may be insufficient to find the few 'islands' of danger in the large space of acceptable safety.

More information could be extracted from simulation models by the aggressive use of fine-grained risk indicators (such as "amount of time that Aircraft1 is closer than minimum separation to another aircraft"). Similarly, beyond the range of simple (failure/no failure) models there is great scope for varying degrees for reduced performance and for subtle gradations of policy or behaviour (such as minimum separation distance or an 'aggression' bias).

The use of such measures and deviations could allow the generation of highly-detailed 'risk landscapes' in which smooth contours would lead towards the regions of lowest and highest risk, potentially allowing the gradients to be followed to find the precise parameter combinations that would lead to accidents. Defined well, such landscapes could make advanced mathematical modelling techniques highly applicable to the hazard analysis problem.

8.2.4 Guidance on Using an Explicit Hierarchy of Models

The work presented in this thesis has concentrated on the production and use of a single simulation model which is used for all analysis. Section 5.8 raised the idea of using higher-fidelity simulation for evaluating derived hazards. This idea could be extended to posit a hierarchy of increasingly detailed models.

The highest level would be paper models that were used to brainstorm for concerns, followed by a low-fidelity constructive simulation (as described in this thesis) to sweep out as much of the space as possible. A corresponding higher-fidelity simulation would allow hypotheses from the low-fidelity model to be quickly explored in more detail and validated. An interactive, multi-user synthetic environment could provide the lowest level of model, for the most realistic analysis at the highest expense.

Dewar discusses a similar approach in [56] under the heading of 'hierarchical analysis' and notes that such multilevel modelling can proceed by identifying concerns for the lower levels based on the factors that appear to be important in the higher level models. For example, Dewar suggests that if the results of a high-level model appear to be sensitive to the range at which targets are detected, then a lower-level model could be created that simulated radar performance in some detail. Such an approach could allow the progressive investigation of the causes of an identified hazard.

8.2.5 Improved Machine Learning Tools

As noted in Section 5.6.3.3, the decision tree learner used in this thesis is only one of many classes of algorithms described in the machine learning literature. Machine learning is an extremely broad and active field of research, with new algorithms emerging regularly. Such algorithms may provide more efficient, capable and comprehensible rule extraction from the simulation output.

One interesting avenue is to explore the use of 'active learning' algorithms (see Cohn in [167]) that can suggest the training instance (in terms of parameter values and their corresponding result) that would be most valuable to increase the validity of the learned model at the current time, allowing the learning algorithm to direct the run selection and thereby minimise the number of runs that need to be performed.

8.2.6 Improvements to Tracing Tool

It was noted in Chapters 6 and 7 that the literature on agent tracing techniques is not well developed, and there is little guidance available on how to derive rules, and to perform optimal logging, for the generation of high-quality tracer graphs. A possible format is 'rule patterns' that would provide outline rules for the explanation of particular types of behaviour. These would then be instantiated for the particular circumstances of the model under analysis.

One potential avenue is to explore the relationship between the tracing tool's output and Why-Because Graphs [168], particularly the potential for the production of explanations using the 'unevent' and 'process' nodes that are used in Why-Because Analysis.

8.2.7 Industrial Trials and Empirical Studies

Although the approach has been evaluated using credible case studies, further confidence of its applicability could be gained by applying it in larger trials as part of SoS acquisition and development products in real industrial and military environments.

The approach would also be effectively evaluated by empirical trials to assess the benefit that the method provides for engineers. For example, two teams of engineers could be assigned to perform hazard analysis of the same SoS in a fixed time period, one team using the method described in this thesis and the other using a manual method (which would need to be identified or developed for this purpose). The completeness and quality of the hazard analysis achieved by each team could then be compared.

8.2.8 Alternative Run-Selection Heuristics

It was observed in Section 4.9.4 that a heuristic is needed to select the simulation runs to be performed (from the intractably large set formed by all possible combinations of vignettes

and deviations). In this thesis, each deviation was assigned a probability of occurrence in a given simulation run, and these were then combined to give a probability for a particular combination of deviations. This is not, however, the only heuristic that could be used, and there may be others that would be more effective at revealing hazards present in the simulation model. Exploration of alternative heuristics would therefore be valuable.

8.2.9 Effectiveness of the Method on Different Classes of SoS

In order for the method presented in this thesis to be improved, it would be valuable to explore how its effectiveness varies depending on the individual characteristics of the particular SoS being analysed. It may be that the method is highly effective for some classes of SoS, but not very useful for certain other classes. The first step in this study would be to identify a set of characteristics on which SoS differ, and then explore the impact of those on the effectiveness of the method, by means of theoretical and empirical work.

8.2.10 Find Influences as Well as Causes

The work in this thesis has concentrated on revealing conditions that directly cause an accident or incident within the simulated model. In combination with a stochastic modelling approach (see Section 8.2.2, above) analysis could be performed to discover those factors that make a given accident more likely. This would not replace the search for causes, but might provide a valuable complement.

8.2.11 Application of the Method to Open SoS

The work in this thesis assumes a 'closed' SoS where a single central owner can be identified. This is reasonable for many military SoS, but an inaccurate description of, for example, the worldwide controlled airspace system. Such 'open' SoS present a variety of additional challenges, such as the potential for the sudden and unexpected introduction of a new component system, and the difficulty of determining the operating policies and procedures for all the various sub-SoS. Further work is needed on how the method described in this thesis can be effectively applied to such systems.

8.2.12 Application of Safety and Hazard Analysis to the Method Itself

One concern in any hazard analysis technique is that the method may have a systematic bias that prevents it from discovering certain kinds of hazards. Similarly, any model used for analysis may have properties that lead to such bias. It would be valuable to explore how conventional hazard analysis techniques could be applied to the process presented here, or to a given model developed using the approach, in order to discover such biases. This would then allow improvements to the process or the model, and allow complementary techniques and models to

be selected in order to make up the deficiency.

8.3 Overall Conclusions

Assessing and ensuring the safety of Systems of Systems is difficult, and given current ambitions (in terms of the size of such systems, along with their level of integration and automation) this difficulty is only likely to increase. There is a paucity of safety engineering techniques that scale effectively to the unique safety challenges posed by modern SoS concepts.

Although multi-agent simulation has been applied to SoS in the past, previous attempts have lacked rigorous modelling methods and provided little help to an analyst who seeks to understand how and why the SoS exhibits the hazards that it does. Indeed, there are no SoS hazard analysis approaches extant in the literature at this time that address the requirements presented in this thesis. The method presented in this thesis goes some way towards addressing these problems, and opens up a new area of hazard analysis research that has the potential to aid greatly in the development of safe and effective SoS.

Appendix A

Despotou's MODAF Model of the AGO SoS

This appendix reproduces the MODAF model presented by Despotou in [89] which was used for the running case study introduced in Section 3.8.

For the purposes of this thesis, the information provided by products OV-1a and 1b can be taken to be that given in the description of the system presented in Section 3.8. These products will therefore not be presented here.

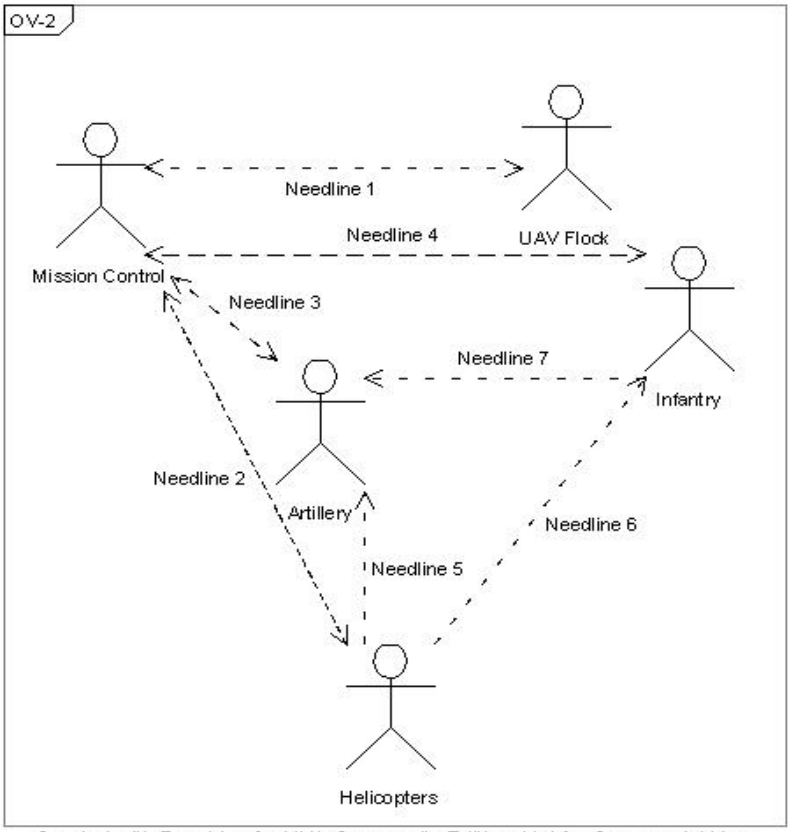

Figure A.1: OV-2 Operational Node Relationships Description

Needline ID	Information Exchange	Producer	Consumer
1	Flight path	Mission Control	UAV Flock
	(Fuse) sensor data	UAV Flock	Mission control
2	Target Area	Mission Control	Helicopters
	(Fuse) sensor data	Helicopters	Mission control
3	Target area	Mission control	Artillery
4	Target area	Mission Control	Infantry
	Map	Mission Control	Infantry
	(Fuse) feedback	Infantry	Mission Control
5	Map	Mission Control	Helicopters
	Target area	Mission Control	Helicopters
	(Fuse) feedback	Helicopters	Mission Control
6	Theatre Support	Helicopters	Infantry
7	Target location	Infantry	Artillery

Figure A.2: Needlines for OV-2

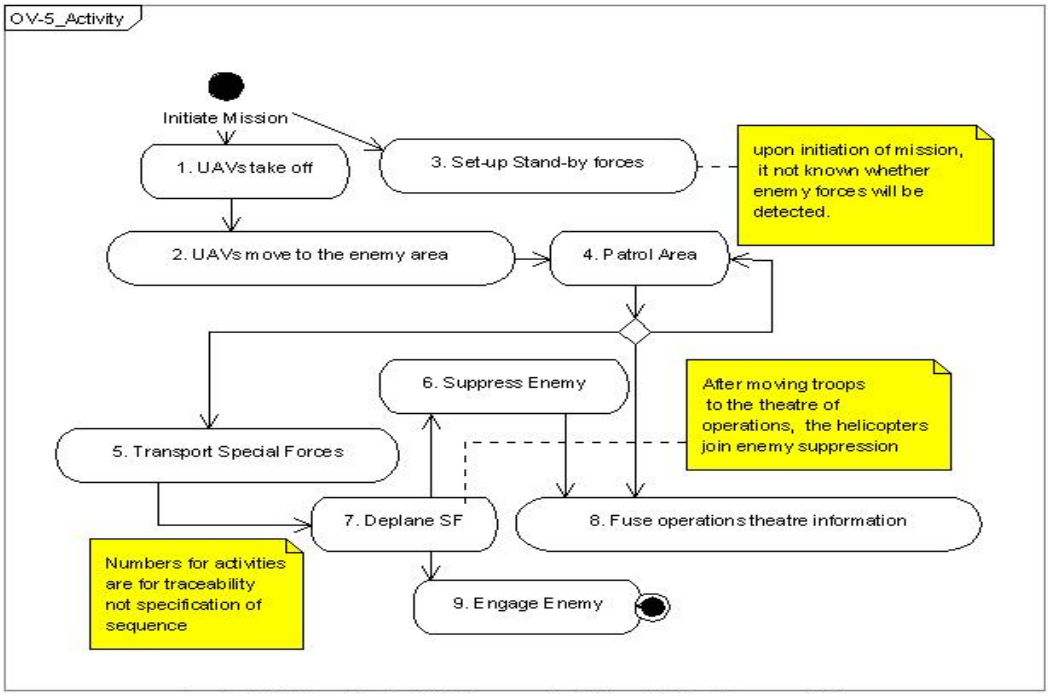

Figure A.3: OV-5 Operational Activity Diagram

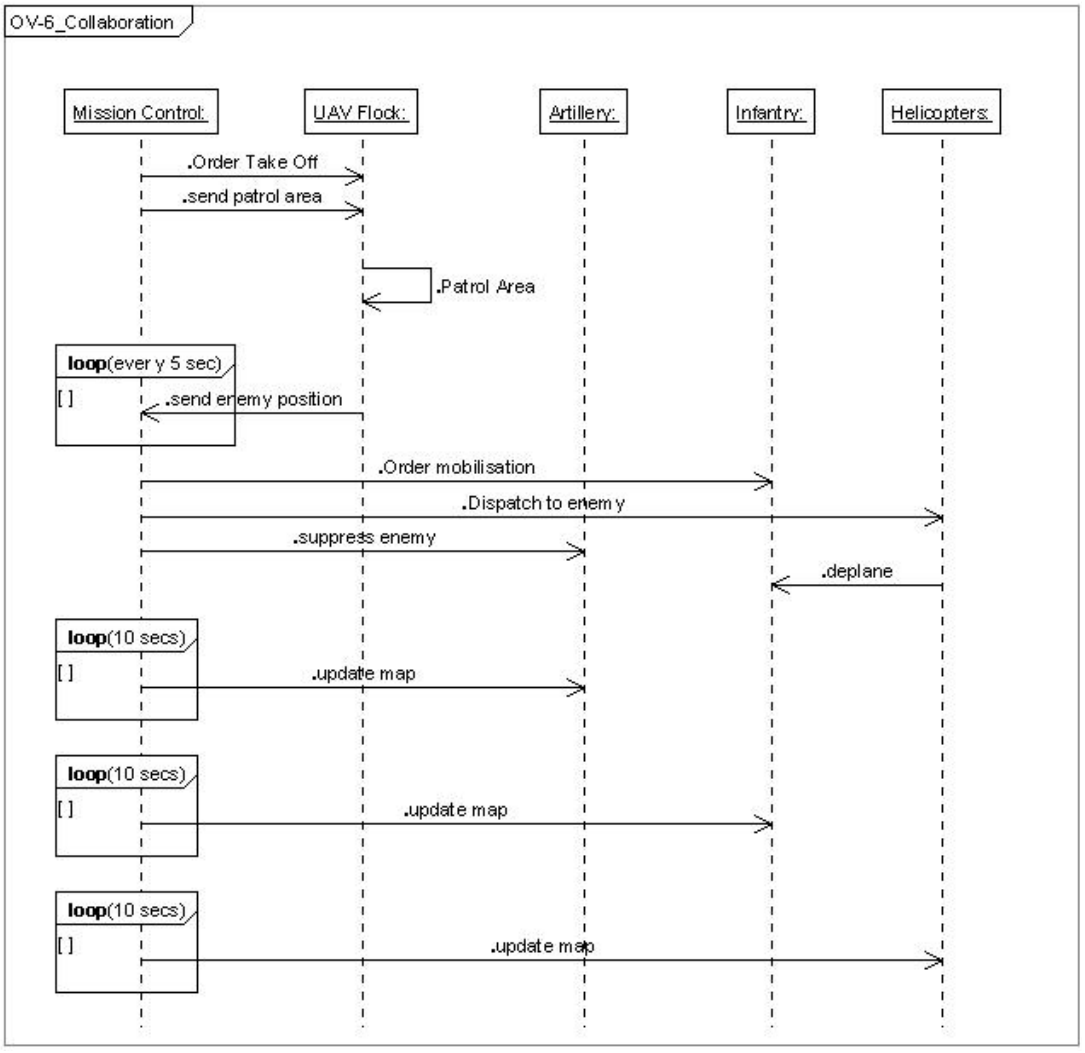

Figure A.4: OV-6c Operational Event-Trace Description

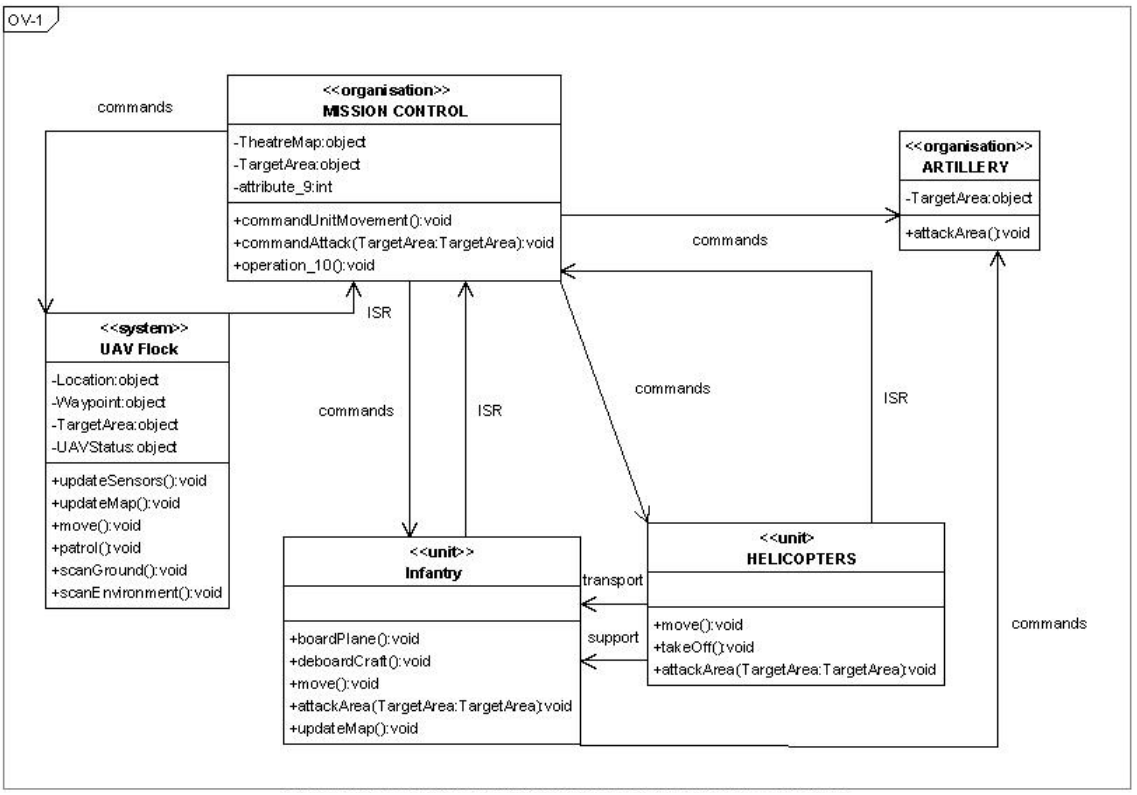

Figure A.5: OV-7 Information Model

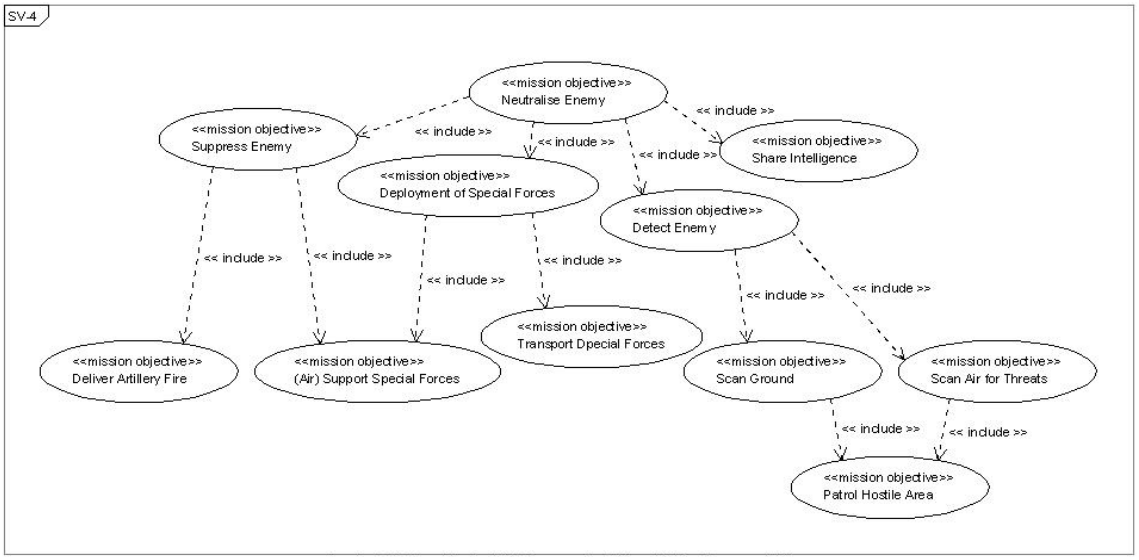

Figure A.6: SV-4 Functionality Description

Activity ID		Mission Control		UAV	Infantry	Helicopters	Artillery
		C4	Comms & Fusion				
1	UAV Take-off	X	X	X			
2	Guidance of UAVs	X	X	X			
3	Set up of forces	X			X	X	X
4	Patrol Enemy Area			X			
5	Transport Special Forces				X	X	
6	Deplane Special Forces				X	X	
7	Suppress Enemy			X		X	X
8	Fuse information		X	X	X	X	
9	Engage Enemy				X		X

Figure A.7: SV-5 Function to Operational Activity Traceability Matrix

Appendix B

Revised AGO MODAF Products

As noted in Section 4.4.1, it was necessary to revise Despotou's original model (as given in Appendix A) before using it for a case study. The revised MODAF products are presented here.

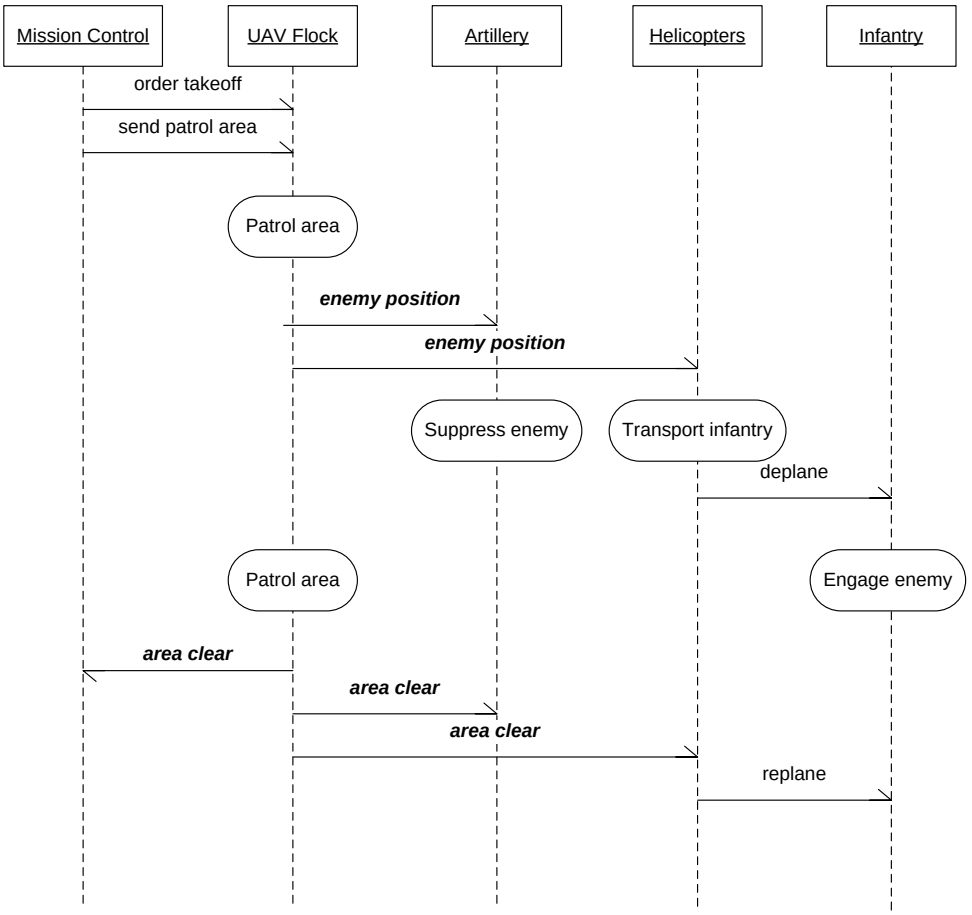

Figure B.1: OV-6c (Operational Event Trace) for AGO SoS

The interactions shown in Figure B.1 with ***bold italic*** names will actually be performed by a continuous (or, at least, very high frequency) data fusion mechanism rather than by irregular explicit messaging, but are shown here as messages to make the sequence diagram intelligible.

Needline ID	Information Exchange	Producer	Consumer
1	Flight path	Mission Control	UAV Flock
2	Target area	Mission Control	Helicopters
3	Target status	UAVs	Helicopters
4	Target area	UAVs	Artillery
5	Target status	UAVs	Artillery
6	Tactical support	Helicopters	Infantry
7	Theater status	UAVs	Mission Control

Table B.1: Needline descriptions for AGO OV-2

It can be observed that needlines 3, 5 and 7 in Figure B.2 (shown as continuous rather than dashed lines) will be met by the data fusion mechanism that provides the COP.

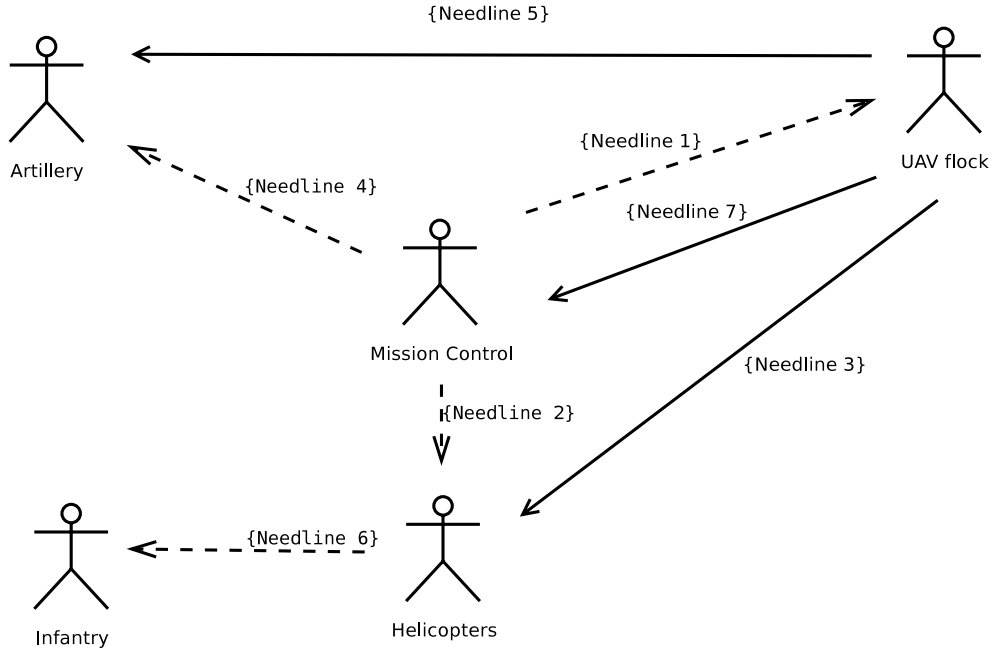

Figure B.2: OV-2 (Operational Node Connectivity Description) for AGO SoS

Appendix C

Prometheus Model of the Adcock System

This appendix provides the Prometheus model that was developed for the case study used in Chapter 6. Most of the content was extracted from the report generated by the Prometheus Design Tool (described by Padgham et al in [97]). Examples of Prometheus modelling outputs were given for the AGO case study in Chapter 4, but the model presented here is larger and more complete. The model presented here provides the design model (see Sections 4.1.2 and 4.8) of the system, which was eventually converted into the Sim8 implementation that was used in the case study.

For clarity, and because of space limitations, the entire output of the tool is not presented here. Rather, all the important diagrams are provided, along with examples of each of the various forms of textual 'descriptor' that accompany them. In the complete design, most diagrams (and many individual diagram elements) have associated descriptors.

C.1 Goal Overview Diagram

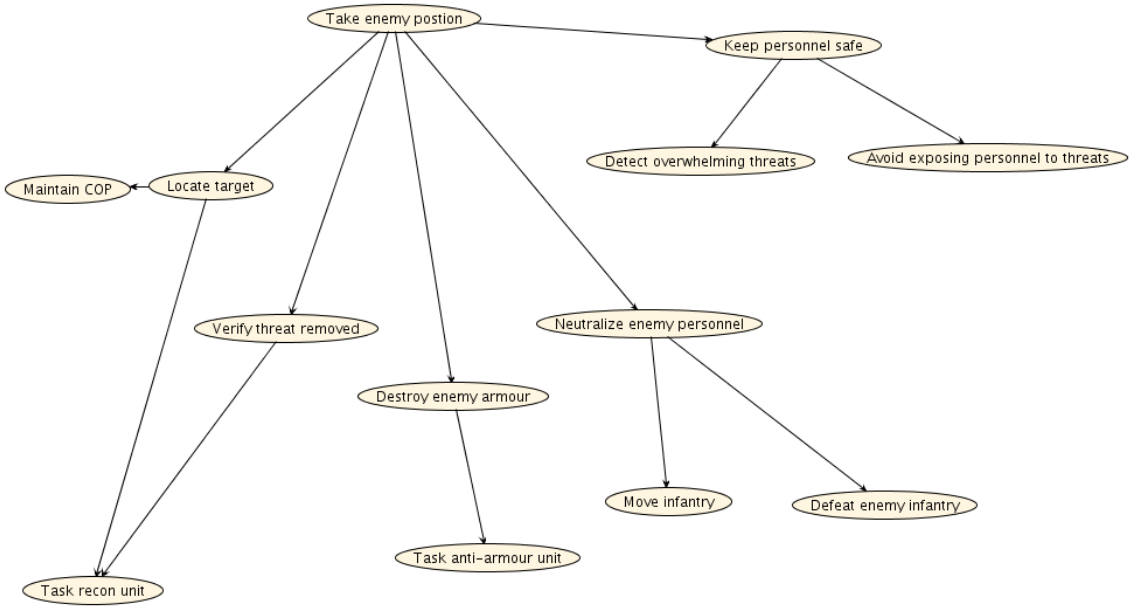

Figure C.1: Goal Overview Diagram

Figure C.1 shows the Goal Overview Diagram, which captures the various goals that apply to SoS at the top level. These goals will need to be satisfied by the combined behaviour of the individual agent models.

Like other Prometheus entities, each goal in the diagram has an associated textual 'descriptor': a small table providing additional detail related to the goal. One example is presented below:

Goal	Locate target
Description	Find suitable enemy targets with the SoS's area of concern, and communicate those targets to the relevant entities within the system
Subgoals	Task recon unit, Maintain COP

C.2 Roles Diagram

Figure C.2 shows the Roles Diagram, which relates the SoS-level goals to a set of 'roles'. Each role draws together a coherent bundle of goals, and at least one agent in the system will later need to be defined that fulfils each role. In the current context, the role diagram is somewhat redundant, because the set of agents is derived from the MODAF model, and not by deciding on the most appropriate breakdown given the roles that need to be fulfilled. It as created, however, because the later stages of the Prometheus process will refer back to elements of this diagram (and the Prometheus Design Tool enforces this).

It can be observed that the Roles Diagram introduces the set of percepts and actions that are

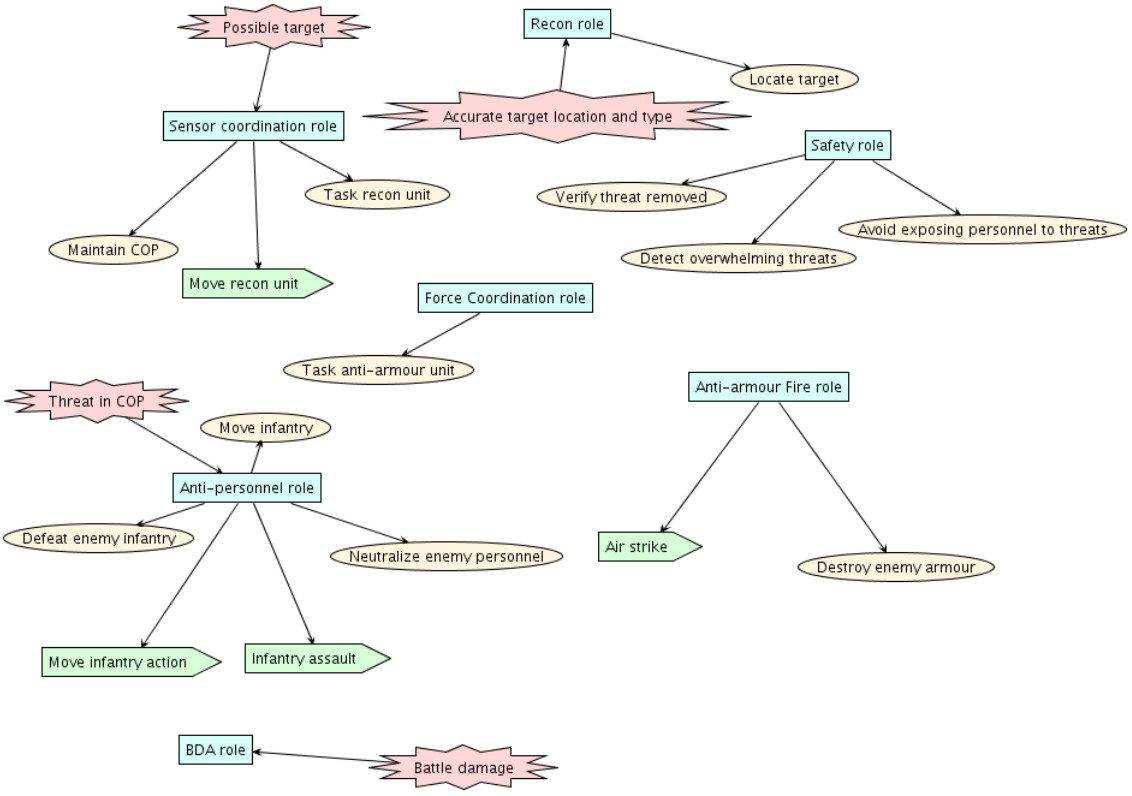

Figure C.2: Roles Diagram

used by each role. The details for these will be presented below.

Each has an associated descriptor:

Role	Sensor coordination
Description	This monitors the environment and, when appropriate, gathers detailed sensor information on important aspects of it.
Percepts	Possible target
Actions	Move recon unit
Information Used	*none*
Information Produced	COP
Goals	Task recon unit, Maintain COP

C.3 Scenario

As part of the Prometheus process, the MODAF scenario presented in Section 6.2.3 was transformed into a more detailed scenario that incorporated additional information.Unlike the MODAF scenario, the Prometheus scenario shows the perceptions that the agents make, the goals that they adopt, and the actions that they perform. Since the set of agents has not been explicitly defined at this point in the Prometheus process, the scenario is expressed in terms of Roles.

#	Type	Name	Role	Description
1	Percept	Possible target	Sensor coordination role	Infantry report possible enemy positions nearby
2	Goal	Locate target	Sensor coordination role	Turn vague position report into a precise one
3	Goal	Task recon unit	Sensor coordination role	Send out something to have a closer look
4	Percept	Accurate target location and type	Recon role	The precise coordinates of the enemy position, and the number and type of units (infantry, armour, etc) stationed there
5	Goal	Maintain COP	Sensor coordination role	Put the new information into the COP so that everyone can see it
6	Goal	Keep personnel safe	Safety role	Protect infantry from enemy that they can't handle on their own (e.g. armour)
7	Goal	Detect overwhelming threats	Safety role	Look in the COP to see if there's anything that the infantry won't be able to deal with
8	Goal	Avoid exposing personnel to threats	Safety role	Find some way to deal with the enemy armour visible in the COP (in this case, by calling for UCAV support)
9	Goal	Destroy enemy armour	Anti-armour Fire role	Bring appropriate weapons to bear on the enemy armour
10	Action	Air Strike	Anti-armour Fire Role	Deliver an air strike against the enemy armour
11	Goal	Keep personnel safe	Safety role	Now check that the armour really was dealt with
12	Goal	Verify threat removed	BDA Role	Check that the enemy armour is now disabled
13	Goal	Task recon unit	Sensor coordination role	Send out something to look at the target of the air strike
14	Percept	Battle damage	BDA Role	Find out effect of air strike on enemy armour effectiveness
15	Goal	Maintain COP	Sensor coordination role	Make battle damage data available to whole system
16	Goal	Neutralise enemy personnel	Anti-personnel role	Want to kill or capture enemy personnel

17	Goal	Move infantry	Anti-personnel role	Move the infantry to the enemy-held location
18	Action	Move infantry action	Anti-personnel role	Infantry uses vehicle and foot movement to reach the enemy location
19	Action	Infantry assault	Anti-personnel role	Infantry assault the enemy position on the ground

C.4 System Overview Diagram

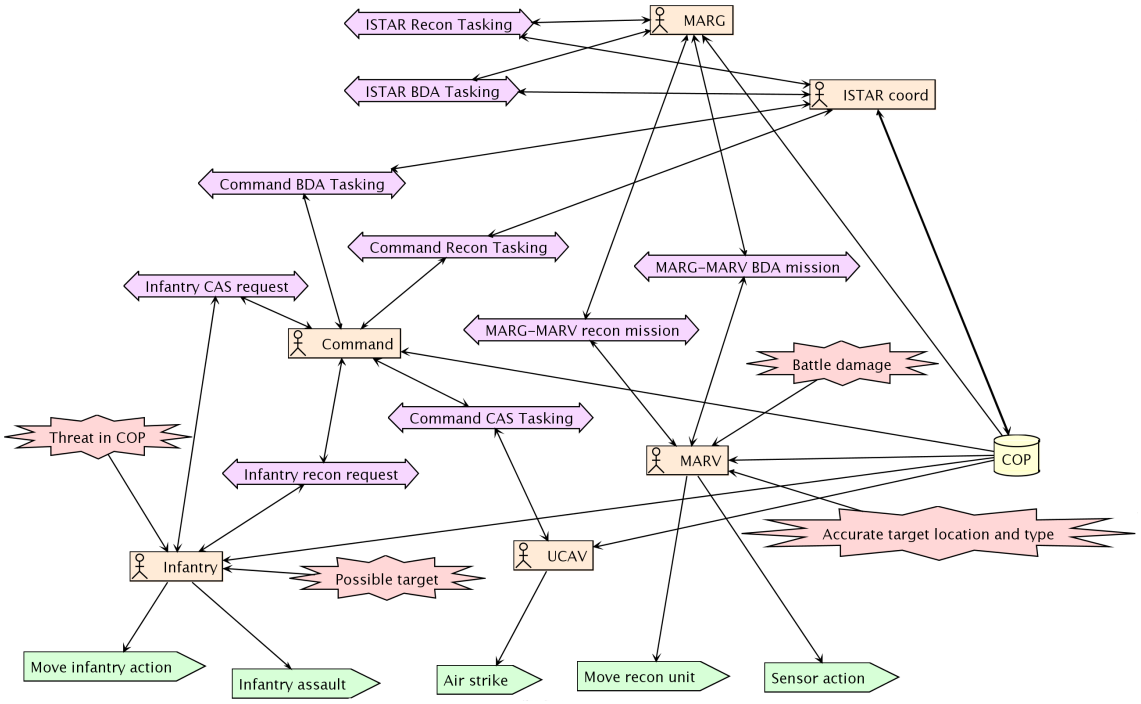

Figure C.3: System Overview Diagram

Figure C.3 shows the System Overview Diagram, which shows the set of agents in the SoS, the actions and percepts they use and the interactions between them. It is closely related to MODAF product OV-2, although it can be observed that, unlike MODAF, all Prometheus artefacts are committed to this set of agents from this point on (see Section 2.6.4 for discussion of this issue with MODAF). In this example, all interaction is via defined protocols (see Section C.12, so individual messages are not featured).

It can be observed that the diagram depicts the Common Operational Picture (COP) as a global data store. With the agents representing geographically distributed entities, there will obviously need to be communication between them in order to achieve this. All of the agents, however, share a common method of exchanging this information, and the Prometheus notation provides

no way to represent this succinctly. The COP is therefore represented in the Design model as a data store, and will need to be expanded during implementation. This is a modelling decision for simplicity and clarity — the details of interaction with the COP could be represented by explicit messaging, but it would be cumbersome and add little to the model at this stage.

C.5 Agents

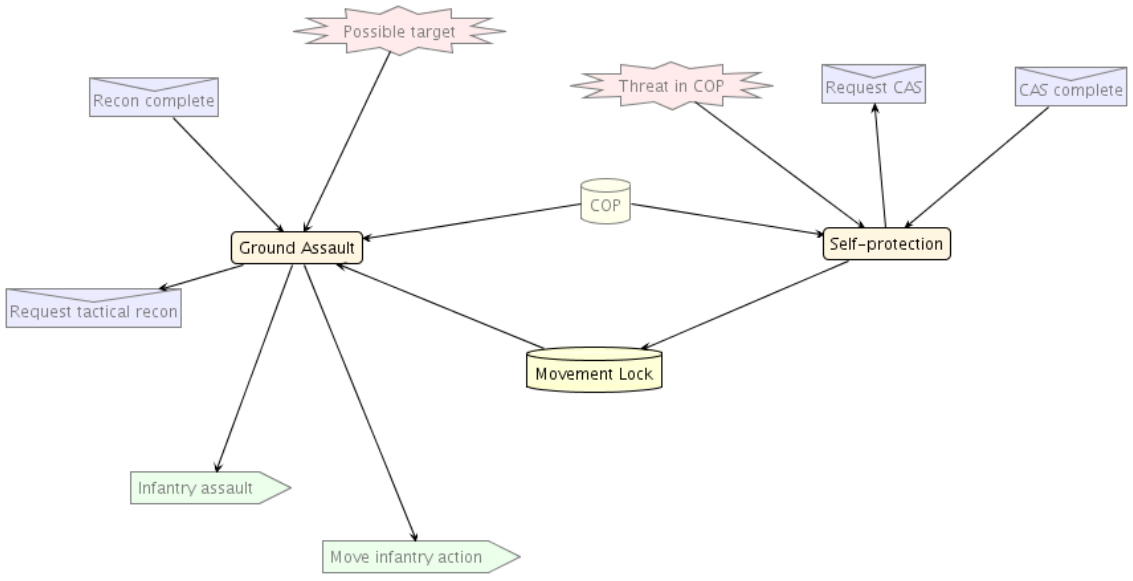

Figure C.4: Agent Overview Diagram for the 'Infantry' Agent

Figure C.4 shows the Agent Overview Diagram for the 'Infantry' agent. This shows how the agent is broken down into various 'capabilities', and how those capabilities interact through both global and local shared data. Each capability is shown with the actions, percepts and messages it uses.

The descriptor of the infantry agent is as follows:

Agent	Infantry
Description	Platoon-sized mechanised infantry unit (approx 30 men, lead by a Lieutenant, in light-armoured vehicles)
Cardinality Minimum	1
Cardinality Maximum	*n/a*
Lifetime	indefinite
Demise	when destroyed
Incoming Messages	Recon complete: Command → Infantry, CAS complete: Infantry → Command
Outgoing Messages	Request tactical recon: Infantry → Command, Request CAS: Infantry → Command
Internal Messages	*none*
Percepts	Possible target, Threat in COP
Actions	Infantry assault, Move infantry
Uses Data	COP
Produces Data	*none*
Internal Data	Movement Lock
Goals	Task anti-armour unit, Move infantry, Defeat enemy infantry, Detect overwhelming threats, Avoid exposing personnel to threats, Verify threat removed, Neutralize enemy personnel
Roles	Anti-personnel role, Safety role
Protocols	Infantry recon request, Infantry CAS request
Included Plans	*none*
Included Capabilities	Ground Assault, Self-protection

Figures C.5 through C.9 show the agent overview diagrams for the other agents in the system.

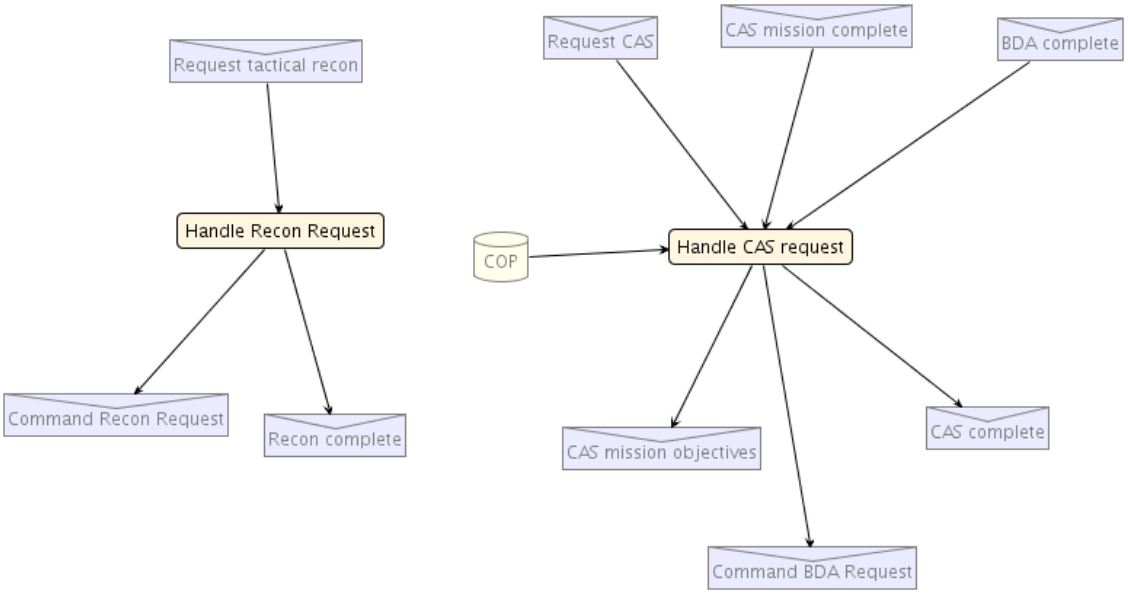

Figure C.5: Agent Overview Diagram for the 'Command' Agent

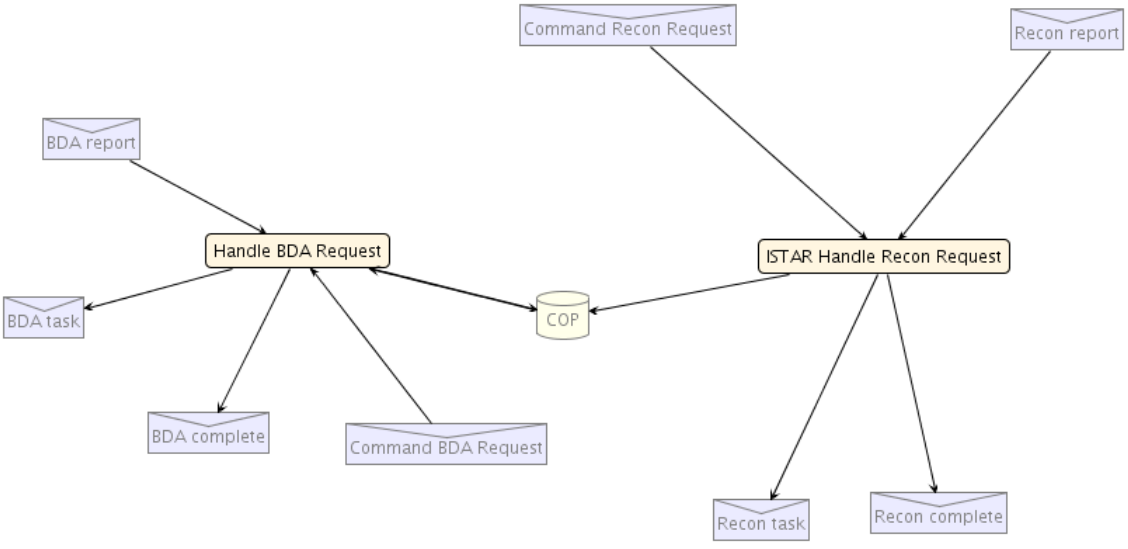

Figure C.6: Agent Overview Diagram for the 'ISTAR coord' Agent

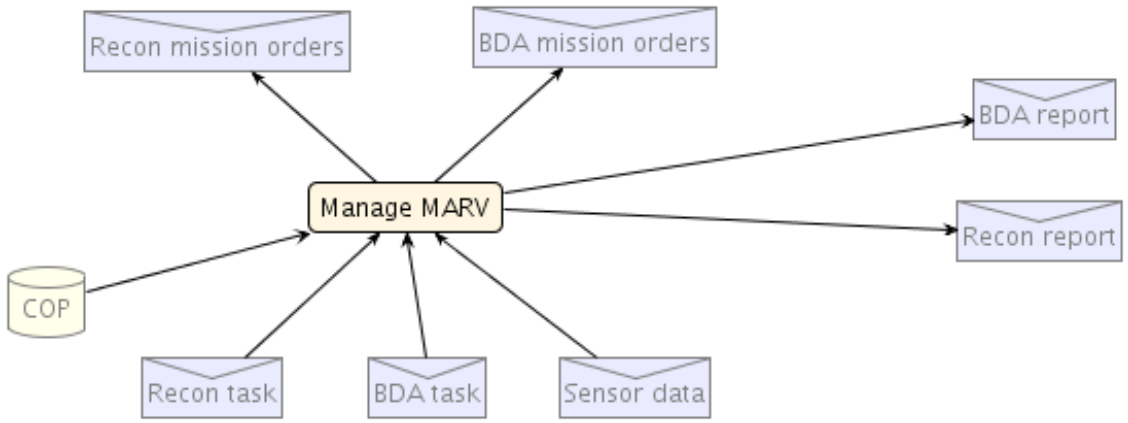

Figure C.7: Agent Overview Diagram for the 'MARG' Agent

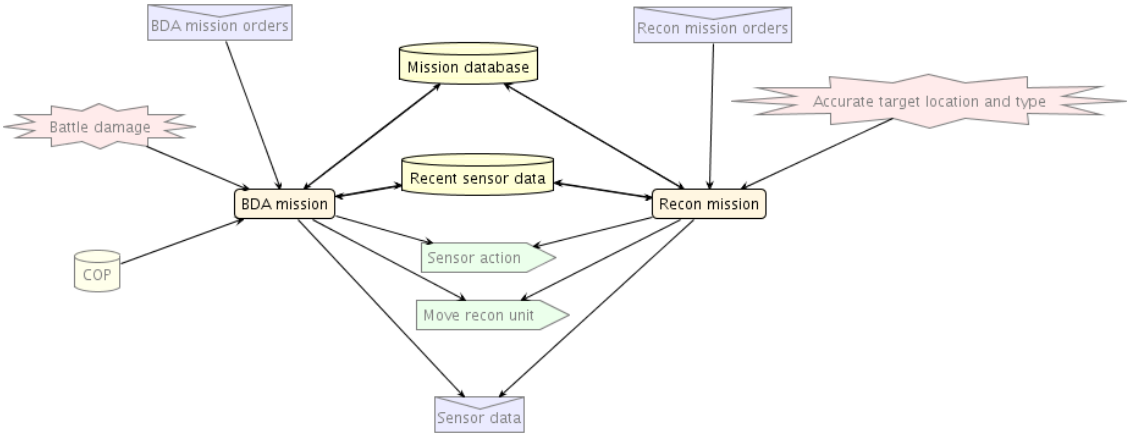

Figure C.8: Agent Overview Diagram for the 'MARV' Agent

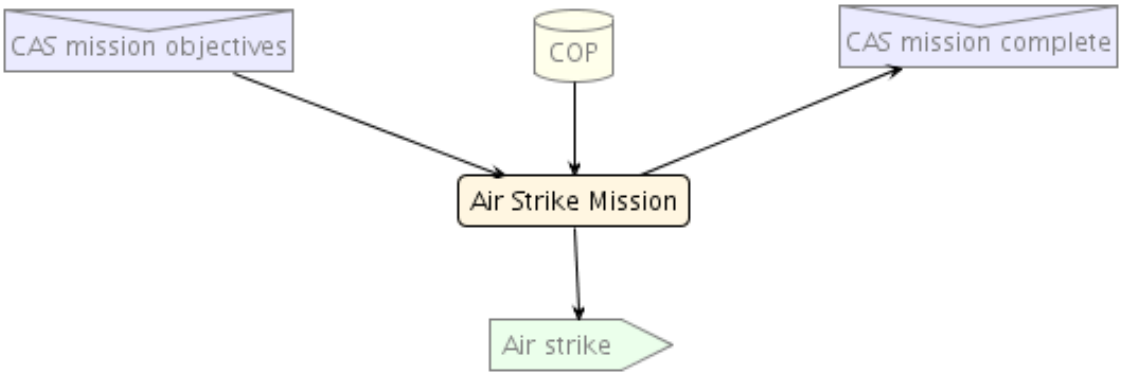

Figure C.9: Agent Overview Diagram for the 'UCAV' Agent

C.6 Capabilities

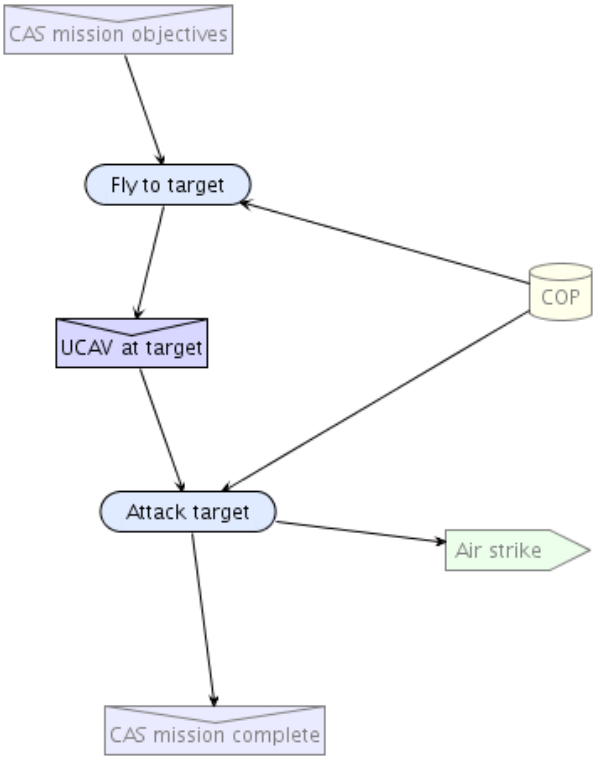

Figure C.10: Agent Overview Diagram for the 'Air Strike Mission' Capability

Figure C.10 shows the Capability Diagram for the 'Air Strike Mission Capability'. This shows how the capability is composed of plans (including the interaction between plans by means of hidden messages) and how the plans relate to messages, actions and data sources.

Capability	Air Strike
Description	The UCAV is able to conduct air strikes on specified ground co-ordinates, destroying or damaging any entities at that location.
Goals	*none*
Processes	*n/a*
Protocols	*none*
Incoming Messages	CAS mission objectives: Command → UCAV, UCAV at target: Fly to target → Attack target
Outgoing Messages	CAS mission complete: UCAV → Command, UCAV at target: Fly to target → Attack target
Internal Messages	UCAV at target: Fly to target → Attack target
Percepts	*none*
Actions	Air strike
Read Data Internal	*none*
Read Data Imported	COP
Written Data Internal	*none*
Written Data Exported	*none*
Included plans	Fly to target, Attack target

Figures C.11 through C.19 are the Capability Diagrams for the other capabilities in the model.

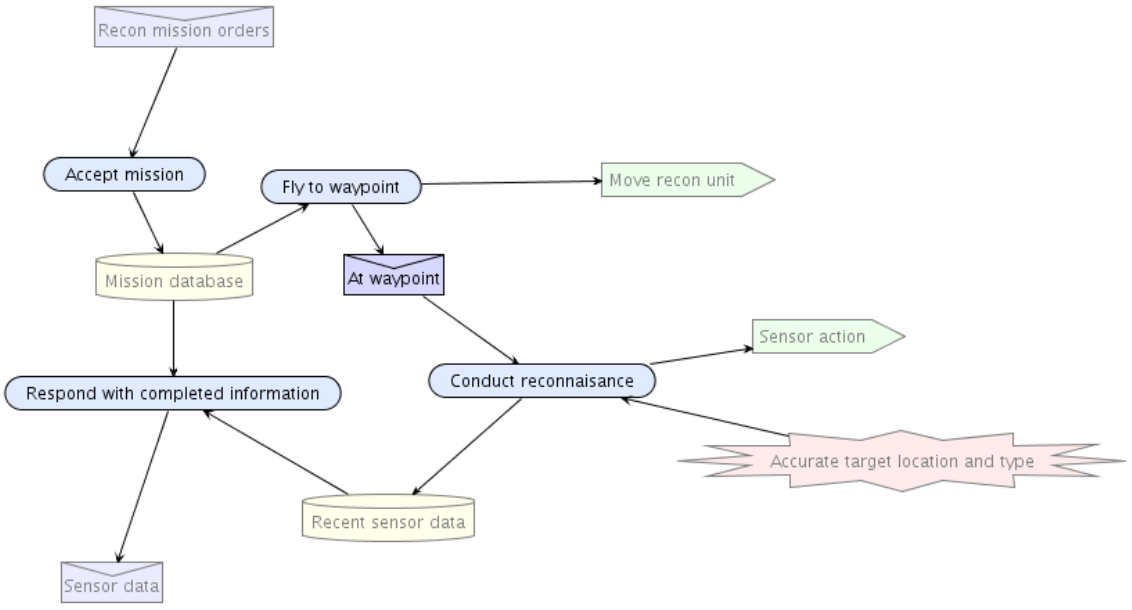

Figure C.11: Capability Diagram for the 'Recon Mission' Capability

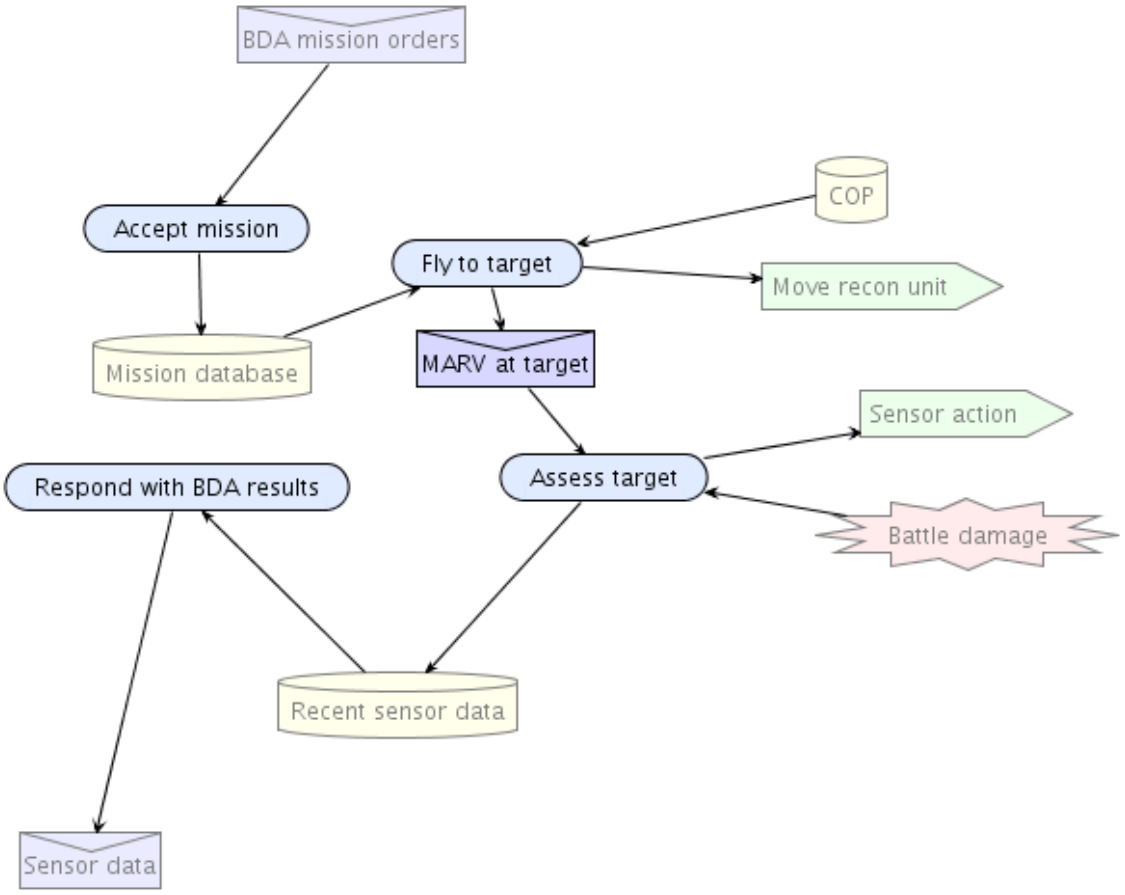

Figure C.12: Capability Overview Diagram for the 'BDA Mission' Capability

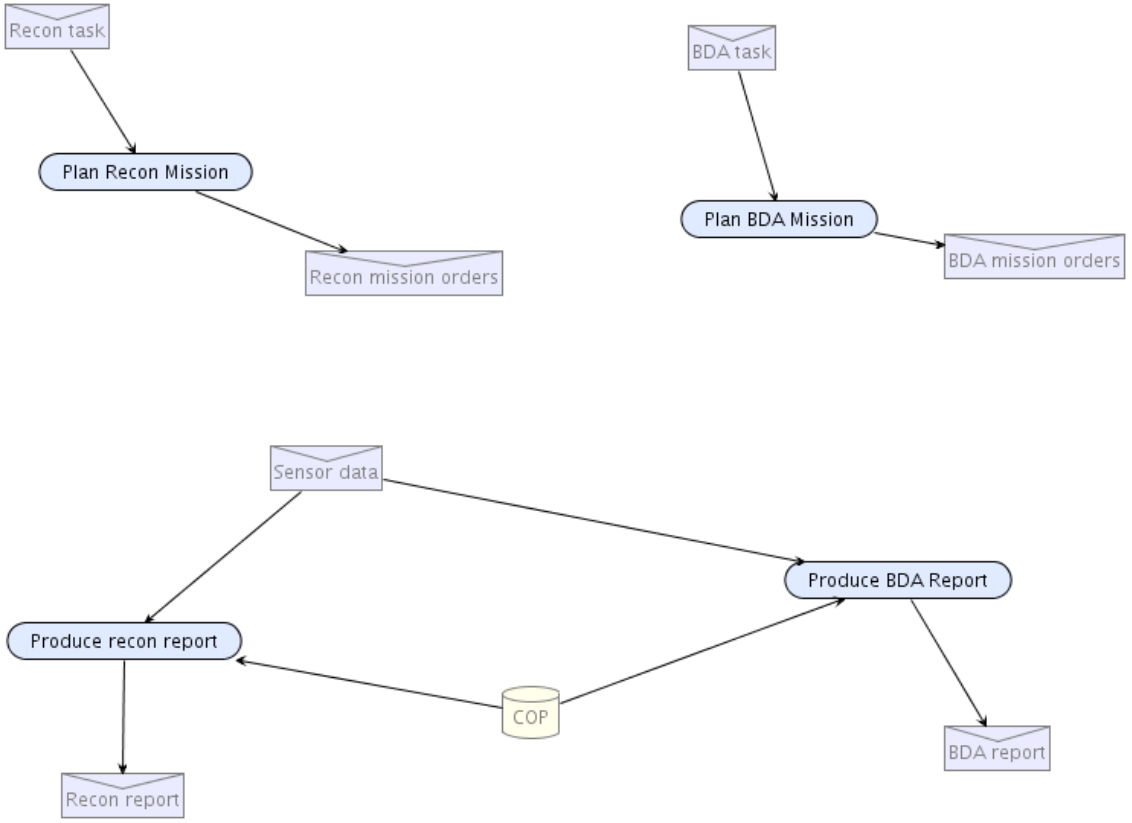

Figure C.13: Capability Overview Diagram for the 'Manage MARV' Capability

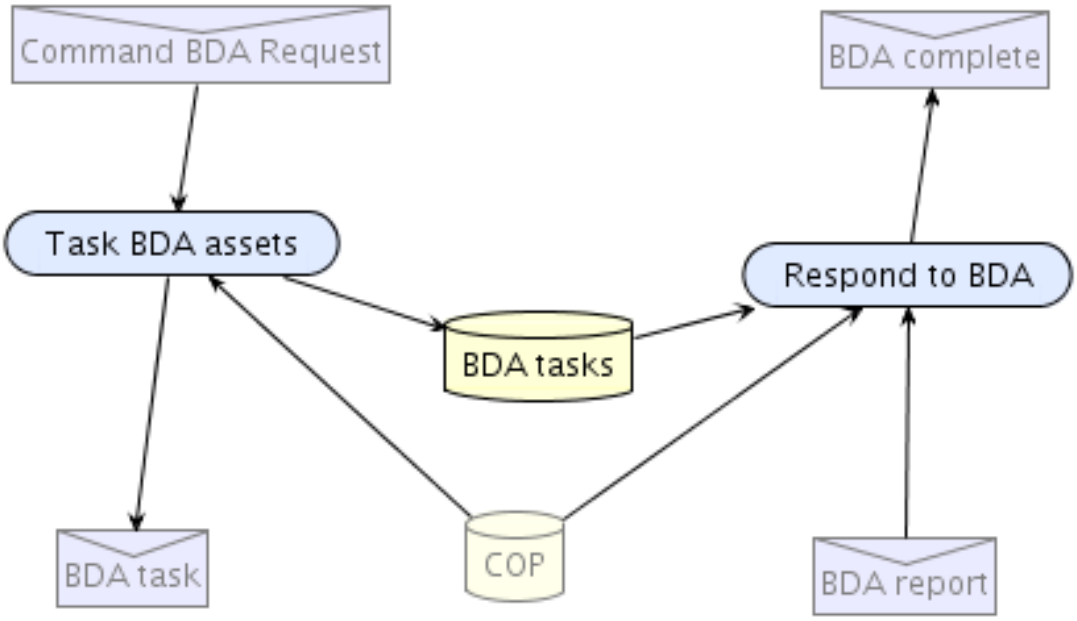

Figure C.14: Capability Overview Diagram for the 'Handle BDA Request' Capability

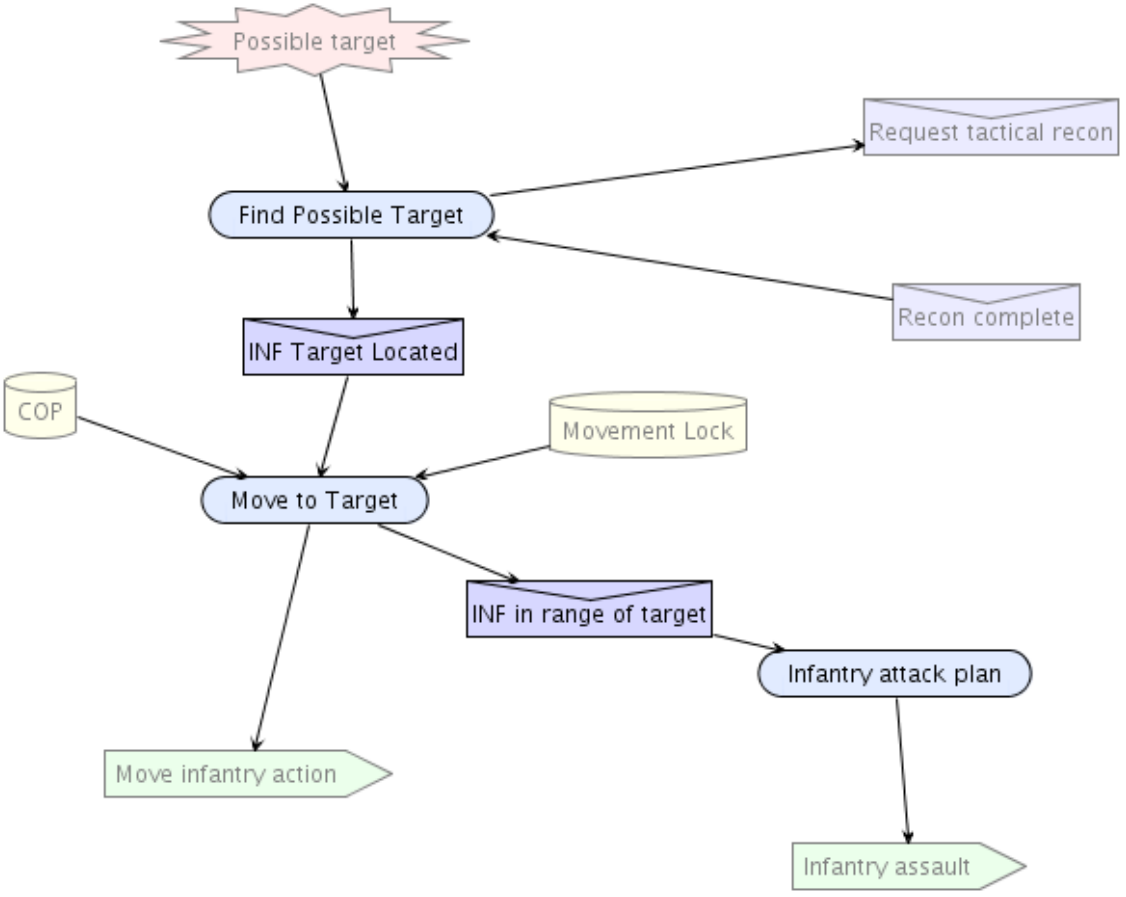

Figure C.15: Capability Overview Diagram for the ‘Ground Assault’ Capability

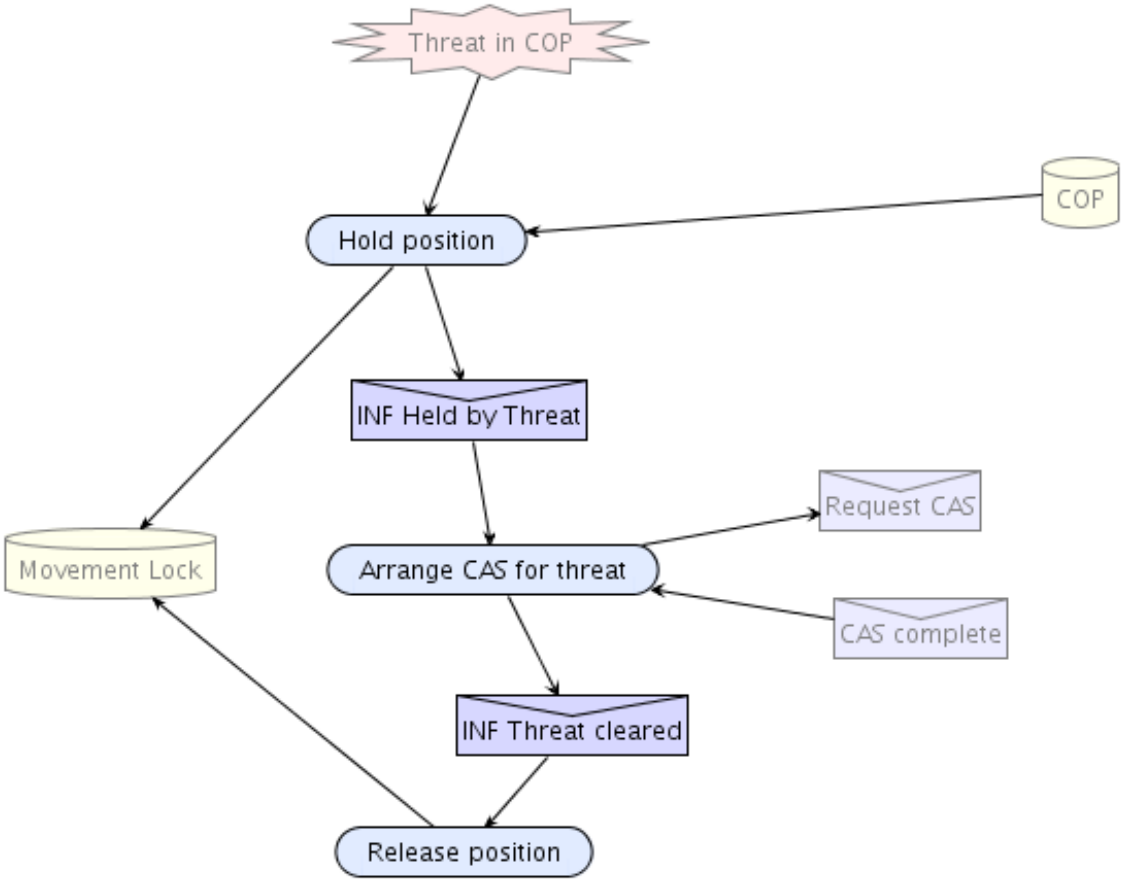

Figure C.16: Capability Overview Diagram for the 'Self-protection' Capability

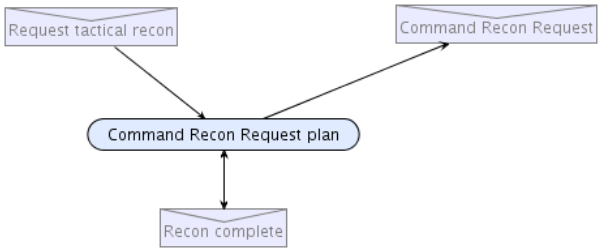

Figure C.17: Capability Overview Diagram for the 'Handle Recon Request' Capability

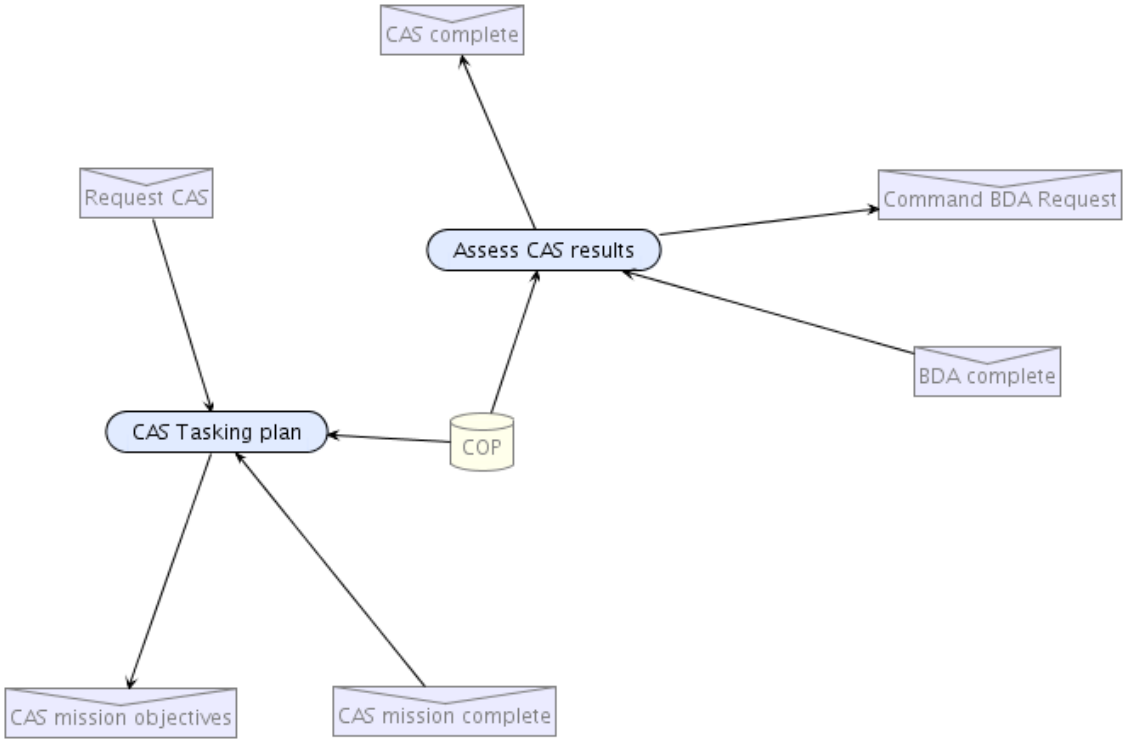

Figure C.18: Capability Overview Diagram for the 'Handle CAS Request' Capability

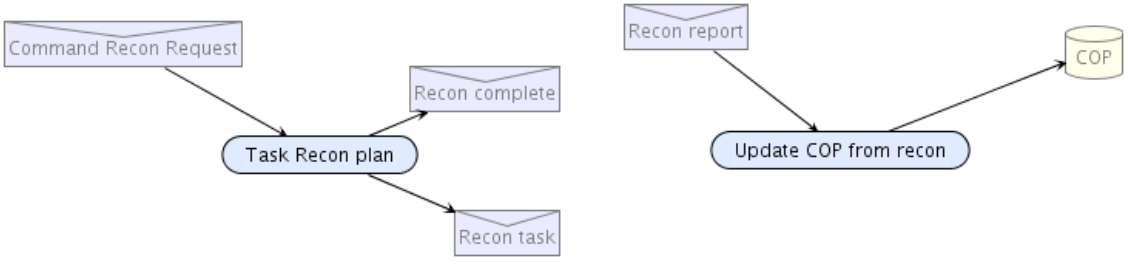

Figure C.19: Capability Overview Diagram for the 'ISTAR Handle Recon Request' Capability

C.7 Plans

The descriptor below is for the plan 'Task Recon Plan', which is used by the ISTAR coordinate to respond to a reconnaissance request by tasking the MARG to arrange MARV reconnaissance of the area in question.

	Task Recon plan
Description	On recept of a reconnaissance request, define an area around the target location given in the request and forward this to the MARG.
Triggers	Recon request received
Context	*n/a*
Incoming Messages	Command Recon Request: Command → ISTAR coord
Outgoing Messages	Recon complete: ISTAR coord → Command, Recon task: ISTAR coord → MARG
Percepts	*none*
Actions	*none*
Used Data	*none*
Produced Data	*none*
Goal	*none*
Failure	*none*
Failure Recovery	*none*
Procedure	• Get request location from message • Define the search area as a radius around request location • Clip the search area by the edge of the SoS's area of concern • Send message to MARG giving the area for reconnaissance • Wait for MARG to send completion message • Send message to Command indicating task complete

C.8 Percepts

Percepts represent observations of the external environment that are made by the agents in the system. There are no percept diagrams, only descriptors. The descriptor below describes the percept 'Accurate target location and type':

Percept	Accurate target location and type
Description	Data gathered by the MARV's sensors
Information Carried	List of entities with force, coords and type (e.g. infantry or armour)
Knowledge Updated	COP, eventually
Source	MARV sensors acting on the environment
Processing	Provided by the environment (although may be manipulated based on MARV sensor faults)
Agents Responding	MARV
Expected Frequency	*unknown*

C.9 Actions

Actions capture the ways by which the agents in the SoS can affect the environment they occupy. The table below is an example of an action descriptor:

Action	Air strike
Description	An attack from the air against a location on the ground that destroys or damages entities at that location
Parameters	Target entities (from COP)
Duration	Minutes
Failure	Entities at target location may not be destroyed
Partial Change	Only some of the entities at the target location may be destroyed. Entities at the location may only be damaged.
Side Effects	*none*

C.10 Messages

Messages are the units of communication between the agents of the SoS. The table below is an example of a message descriptor.

Message	Recon mission orders
Description	Precise orders for a reconnaissance mission, so that it can be followed by the MARV (which is only semi-autonomous and therefore needs detailed instructions).
Distribution	MARG → MARV
Carried Information	List of waypoints to fly to, list of sensors that need to be active

C.11 Data Stores

Data stores express the data (and, by extension, the beliefs) that is held and used by the entities in the SoS. The 'COP' data source, which is used by all the agents in this model, is shown here:

Data	COP
Description	An abstract representation of the Common Operating Picture that is maintained by all the agents in the SoS. For adequate hazard and safety analysis, this abstract view will have to be replaced by an explicit mechanism for maintaining this via communication and data fusion.
Persistent	Yes
External to System	No
Connections	COP written by (ISTAR coord) read by (ISTAR coord, MARG, MARV, UCAV, Command, Infantry
Initialisation	Empty
Produced By	ISTAR coord, Handle BDA Request, Handle Recon Request
Used By	ISTAR coord, MARG, MARV, UCAV, Command, Infantry, Air Strike Mission, BDA mission, Manage MARV, Handle BDA Request, Ground Assault, Handle BDA Request, Handle CAS request, Self-protection

C.12 Protocols

Figure C.20 shows the Protocol Diagram 'Infantry recon request'. It can be seen that this takes the form of a simple request followed by an eventual acknowledgement that the request has been fulfilled. This is a very simple example, but Prometheus allows the full Agent UML protocol notation and thereby permits sophisticated protocols.

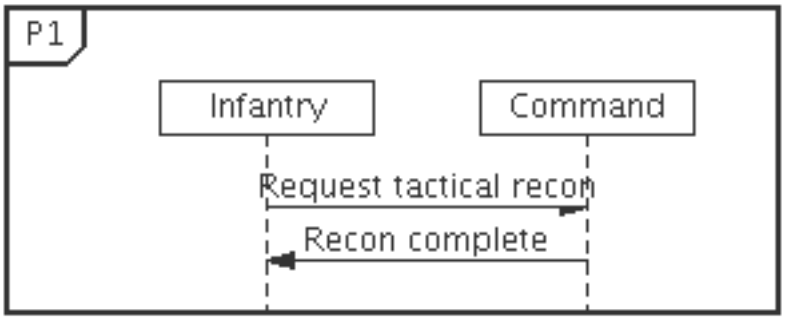

Figure C.20: Protocol Diagram for the 'Infantry Recon Request' Protocol

Protocol	Infantry recon request
Description	Infantry has seen evidence of a potential target or threat (possibly in space, possibly in the COP) and wants accurate reconnaissance information for it.
Included Messages	Request tactical recon: Infantry → Command, Recon complete: Command → Infantry
Agents	Command, Infantry
AUML	• start P1 • agent IF Infantry • agent COM Command • message IF COM Request tactical recon • message COM IF Recon complete • finish

C.13 Observations About the Design Model

It can be seen from the above that an explicit set of SoS component systems has been defined, and that all those component systems have explicit behaviour associated with them. Generalised patterns of interaction have been identified (in contrast with the specific single-sequence interaction scenarios expressed in a MODAF OV-6c, for example) and a scenario has been defined that combines agent internal behaviour, agent communication, and agent external behaviour.

Now that this model has been created, there is a clear path to implementation as an executable simulation model. This path was not obvious given only the original MODAF model. Depending on the target implementation platform (and hence the constraints on the operation model — see Section 4.12) this translation may be straightforward.

The model is provided in terms of an explicit BDI architecture. This provides an easy route to implementation using an existing BDI engine, and supports analysis via agent tracing tools as described in Section 5.7.1. It can be observed, however, that the only explicit goals are associated with the SoS as a whole, not the individual agents. Assumptions will have to be made if these are to be reintroduced at the implementation stage. Similarly, there is some mapping needed between data stores that the Prometheus model uses the and individual BDI 'beliefs'. Despite these limitations, for the analysis described in this thesis the model is an significant improvement on the MODAF model it was derived from.

Appendix D

Deviations for the Adcock System

The tables in this appendix present the agent-level deviations that were defined for the the case study used in Chapter 6, based on the deviation approach defined in Section 4.9. Each table corresponds to one agent (with the first table covering agent 'All' — deviations that are common to all the agents in the system).

Each row gives the specific agent and part that it applies to, the generic deviation that has been used, and any additional details that clarify the specific nature of the deviation (including some details in terms of the operational model that are needed for implementation). A '—' in the details column indicates that the implementation of the generic deviation is straightforward. Generic deviations that are not present represent cases where no specific deviation could be derived for the agent and part in question.

In some cases, the lack of generic deviations in a specific cases follows from the nature of the actual agent or part. For example, the command agent has no actuator deviations because it has no actuators. In others, the absence is due to the way the agent has been modelled. For example, the UCAV must clearly have some sensors in order to carry out its missions, but since they are not directly represented in the model (for example, it doesn't contribute anything back to the COP), no deviations are defined for them.

Table D.1: Deviations applicable to all agents

Agent	Part/Service	Generic Deviation	Details	ID
All	Comms	Can't send	—	A1
		Can't receive	—	A2
		Entity excluded from network	Don't include (as sender or receiver) in the data fusion network	A3
		Duplicate messages sent	Each message twice	A4
		Duplicate messages sent later	10 ticks later (arbitrary)	A5
		Additional bandwidth used	Double	A6
		Delay in sending messages	10 ticks later (arbitrary)	A7
		Delay in receiving/processing messages	10 ticks later (arbitrary)	A8
	SA/Data Fusion	SA does not persist	SA cleared at start of every tick	A9
		Duplication of entity traces	Two of each	A10
		Trace prematurely removed from SA	Trace disappears from SA after 10 ticks	A11
			Max of 2 entities in SA at once	A12
		Trace persists in SA after known to be moved or destroyed	Ignore 'moved' traces, can't tell if something has died.	A13
		SA coordinate system mismatched with peer entities	Offset coordinate system. 1 sq distortions in each cardinal direction.	A14
	Computation / Thinking	Delay in processing	Limited rate of processing new contacts — one per tick	A15

Table D.2: Deviations applicable to Infantry agents

Agent	Part/Service	Generic Deviation	Details	ID
Infantry	Local area sensor	Total loss of sensing	—	IN1
		Reduction of sensor range	To 0 — same square only.	IN2
		Increase of sensor range	To 2.	IN3
		Delay in registering sensor contacts	10 ticks	IN4
		Incorrect identification of contact side/force	Inverted (i.e. RED $\Longleftrightarrow$ BLUE)	IN5
		Incorrect identification of contact entity type	See everything as being generic entity type	IN6
		Incorrect determination of contact location	Displace 1 sq in a cardinal direction	IN7
	Possible target' sensor	Total loss of sensing	—	IN8
		Reduction of sensor range	Drop to half (3)	IN9
		Duplication of contacts	Has to sense each entity twice before it is 'known'.	IN10
		Increase of sensor range	Double (10).	IN11
		Delay in registering sensor contacts	10 ticks	IN12
	Movement actuator	Total loss of function	—	IN13
		Reduction in magnitude of function	Move speed halved (—4)	IN14
		Function provided when not required	Chance of random move each tick	IN15
		Increase in magnitude of function	Double speed (1)	IN16
		Delay in performing function	Movement actually happens 1 tick after it is ordered	IN17

		General loss of precision / control	Chance of move direction (when move ordered) being randomly changed	IN18
	Small arms fire actuator	Total loss of function	Cannot fire	IN19
		Reduction in magnitude of function	Half damage	IN20
		Partial loss of applicability of function	Reduced range – can only hit targets in same square	IN21
		Increase in magnitude of function	Double damage	IN22
		Delay in performing function	Attack on following tick.	IN23
	Find Possible Target plan	Plan step omitted	Drop whole plan	IN24
		Trigger condition not implemented	—	IN25
		Plan step duplicated	Request recon twice	IN26
		Extra trigger condition	Triggered by new enemy in SA	IN27
		Plan step moved earlier in sequence	Resume moving before reply message arrives	IN28
		Substitute entire plan with another	Move To Target	IN29
	Move To Target plan	Plan step omitted	Don't send arrival message	IN30
		Trigger condition not implemented	—	IN31
		Plan step duplicated	Send arrival message twice	IN32
		Plan step moved earlier in sequence	Send arrival message before reaching target location	IN33
		Substitute entire plan with another	Arrange CAS for Threat	IN34
	Infantry Attack plan	Plan step omitted	Don't attack	IN35

		Trigger condition not implemented	—	IN36
		Extra trigger condition	Trigger when any entity in range	IN37
	Arrange CAS for Threat plan	Plan step omitted	Don't lock movement	IN38
			Don't request CAS	IN39
		Trigger condition not implemented	—	IN40
		Plan step duplicated	Request CAS twice	IN41
		Extra trigger condition	Trigger on presence of any enemy, not just armour	IN42
		Plan step moved earlier in sequence	Release move lock before 'CAS complete' received	IN43
		Substitute entire plan with another	Move To Target plan	IN44

Table D.3: Deviations applicable to Command agents

Agent	**Part/Service**	**Generic Deviation**	**Details**	**ID**
Command	Computation / Thinking	Delay in processing	Tasking proceeds at a limited rate	C1
			Tasking delayed by 10 ticks	C2
	Recon plan	Plan step omitted	Don't propagate recon request	C3
			Don't propagate recon completion	C4
		Trigger condition not implemented	—	C5
		Plan step duplicated	Send extra recon request	C6
			Send extra recon completion	C7
		Extra trigger condition	Trigger on CAS request	C8
		Plan step moved earlier in sequence	Send recon complete before receiving completion from ISTAR	C9
		Plan step moved later in sequence	—	C10

		Substitute entire plan with another	Replace with CAS Tasking plan	C11
	CAS Tasking plan	Plan step omitted	Don't send CAS objectives	C12
			Don't perform BDA	C13
		Trigger condition not implemented	—	C14
		Plan step duplicated	Send two CAS requests	C15
		Extra trigger condition	Trigger on recon request	C16
		Plan step moved earlier in sequence	Send completion before tasking UCAV	C17
			Send completion before tasking BDA	C18
		Substitute entire plan with another	Substitute Assess CAS Results	C19
	Assess CAS Results plan	Plan step omitted	Don't send CAS completion	C20
			Don't send BDA request	C21
		Trigger condition not implemented	—	C22
		Plan step duplicated	Send two BDA requests	C23
		Extra trigger condition	Trigger on CAS request (so simultaneous with CAS mission)	C24
		Plan step moved earlier in sequence	Send CAS complete before received BDA complete	C25
		Substitute entire plan with another	Recon plan	C26

Table D.4: Deviations applicable to ISTAR agents

Agent	**Part/Service**	**Generic Deviation**	**Details**	**ID**
ISTAR	Computation / Thinking	Delay in processing	Tasking proceeds at a limited rate	IS1
			Tasking delayed by 10 ticks	IS2
	Task Recon plan	Plan step omitted	Don't send recon task	IS3
			Don't send recon complete	IS4

		Plan step duplicated	Send two recon requests	IS5
		Extra trigger condition	On receipt of BDA request	IS6
		Plan step moved earlier in sequence	Send recon complete before receive completion message from MARG	IS7
		Substitute entire plan with another	Task BDA Assets	IS8
	Update COP from Recon plan	Plan step omitted	Don't update the COP	IS9
		Late	Plan takes 10 ticks to perform update	IS10
	Task BDA Assets plan	Plan step omitted	Don't send BDA task	IS11
			Don't send BDA complete	IS12
		Plan step duplicated	Send two BDA requests	IS13
		Extra trigger condition	On receipt of recon request	IS14
		Plan step moved earlier in sequence	Send BDA complete before receive completion message from MARG	IS15
		Substitute entire plan with another	Task Recon	IS16
	Respond to BDA plan	Plan step omitted	Don't update the COP	IS17
		Late	Plan takes 10 ticks to perform update	IS18

Table D.5: Deviations applicable to MARG agents

Agent	**Part/Service**	**Generic Deviation**	**Details**	**ID**
MARG	Plan Recon Mission plan	Plan step omitted	Don't task MARV	MG1
			Don't plan a route, just send one target waypoint	MG2
		Trigger condition not implemented	—	MG3

		Plan step duplicated	Task MARV twice	MG4
		Extra trigger condition	On receipt of BDA task	MG5
		Plan takes more thinking/processing time	16 (instead of default 2)	MG6
		Substitute entire plan with another	Plan BDA Mission	MG7
	Produce Recon Report plan	Plan step omitted	Don't produce report	MG8
		Plan step duplicated	Send two reports	MG9
		Subsitute entire plan with another	Produce BDA Report	MG10
	Plan BDA Mission plan	Plan step omitted	Don't task MARV	MG11
		Trigger condition not implemented	—	MG12
		Plan step duplicated	Task MARV twice	MG13
		Extra trigger condition	On receipt of Recon task	MG14
		Plan takes more thinking / processing time	10 (instead of default 1)	MG15
		Substitute entire plan with another	Plan Recon Mission	MG16
	Produce BDA Report plan	Plan step omitted	Don't produce report	MG17
		Plan step duplicated	Send two reports	MG18
		Subsitute entire plan with another	Produce Recon Report	MG19

Table D.6: Deviations applicable to MARV agents

Agent	Part/Service	Generic Deviation	Details	ID
MARV	Local Airspace sensor	Total loss of sensing	—	MV1
		Increase of sensor range	Entities at range 2 report as being at range 1	MV2
		Duplication of contacts	Reported as also being in a random adjacent square	MV3
		Wholly imaginary contacts	Chance of random contact each tick	MV4
		Delay in registering sensor contacts	10 ticks	MV5
		Incorrect determination of contact location	Displace 1 sq in a cardinal direction	MV6
	Ground sensor	Total loss of sensing		MV7
		Increase of sensor range	Contacts detected over a larger area	MV8
		Duplication of contacts	Reported as also being in a random adjacent square	MV9
		Wholly imaginary contacts	Chance of random contact each tick	MV10
		Delay in registering sensor contacts	10 ticks	MV11
		Incorrect identification of contact side/force	Inverted (i.e. RED $\Longleftrightarrow$ BLUE)	MV12
		Incorrect identification of contact entity type	See everything as being generic entity type	MV13
		Incorrect determination of contact location	Displace 1 sq in a cardinal direction	MV14
	Flight actuator	Total loss of function	—	MV15
		Reduction in magnitude of function	Move speed halved	MV16

		Function provided when not required	Chance of random move each tick	MV17
		Increase in magnitude of function	Double speed	MV18
		Delay in performing function	Movement actually happens 1 tick after it is ordered	MV19
		General loss of precision / control	Chance of move direction (when move ordered) being randomly changed	MV20
	Accept Mission plan	Plan step omitted	Don't fly to waypoints	MV21
			Don't send report	MV22
		Trigger condition not implemented	—	MV23
		Plan step duplicated	Send report twice	MV24
			Repeat waypoints twice before sending report	MV25
		Substitute entire plan with another	Plan BDA Mission	MV26
	Fly to Waypoint plan	Plan step duplicated	Add intermediate waypoint	MV27
		Plan step moved earlier in sequence	Report arrival before reaching waypoint	MV28
		Plan step moved later in sequence	Report arrival after some significant distance past waypoint	MV29
	Conduct Recon plan	Plan step omitted	Don't perform sensing	MV30
		Trigger condition not implemented	—	MV31
		Substitute entire plan with another	Plan BDA Mission	MV32
	Respond with Completed Recon plan	Plans takes more thinking/processing time	Delay in sending results	MV33
		Plan step omitted	Don't send data	MV34

	Respond with BDA Results plan	Plans takes more thinking/processing time	Delay in sending results	MV35
		Plan step omitted	Don't send data	MV36

Table D.7: Deviations applicable to UCAV agents

Agent	**Part/Service**	**Generic Deviation**	**Details**	**ID**
UCAV	Flight actuator	Total loss of function	—	U1
		Reduction in magnitude of function	Move speed halved	U2
		Function provided when not required	Chance of random move each tick	U3
		Increase in magnitude of function	Double speed	U4
		Delay in performing function	Movement actually happens 1 tick after it is ordered	U5
		General loss of precision / control	Chance of move direction (when move ordered) being randomly changed	U6
	Airstrike actuator	Total loss of function	Cannot strike	U7
		Reduction in magnitude of function	Half damage	U8
			Cannot damage armoured targets	U9
		Increase in magnitude of function	Double damage	U10
		Delay in performing function	Attack on following tick	U11
		General loss of precision / control	Strike hits target square *and* surrounding squares	U12
	Fly to Target plan	Plan step omitted	Don't report arrival	U13
		Plan step duplicated	Add intermediate waypoint	U14
		Plan step moved earlier in sequence	Report arrival before reaching target	U15

		Plan step moved later in sequence	Report arrival after some significant distance past target	U16
	Attack target plan	Plan step omitted	Don't deliver strike	U17
		Plan step duplicated	Strike twice	U18

Appendix E

Log Output Extract from Adcock Model

This appendix contains an example of the log output from the Adcock model (as discussed in Chapter 6). As the original log was voluminous, much of the repetition has been elided here — e.g. repeated movements relating to the same high-level agent action have been removed. This data was *not* removed from the logs that were used for automated analysis.

```
belief,inf1, waypoint: int3d[60,5,0],0,0
percept,inf1,newentitynearbypercept,10,0
belief,inf1,new entity nearby,10,0
goal,inf1,find nearby entity,10,0
plan,inf1,get local recon from command,10,0
message,inf1,command: reconrequestmessage,10,0
goal,command, fulfil recon request,10,0
plan,command, order recon via istar,10,0
message,command,marg: reconrequestmessage,10,0
goal,marg,perform recon around int2d[15,5],10,0
plan,marg,plan recon mission,10,0
message,marg,marv1: reconmissionordersmessage,11,0
belief,marv1, waypoint: int3d[12,2,1],11,0
belief,marv1, waypoint: int3d[12,7,1],11,0
...
belief,marv1, arrived at waypoint int3d[12,2,1],22,0
plan,marv1,conduct recon,22,0
belief,marv1, arrived at waypoint int3d[12,7,1],27,0
plan,marv1,conduct recon,27,0
...
belief,marv1, arrived at waypoint int3d[22,7,1],67,0
plan,marv1,conduct recon,67,0
percept,marv1,tracepercept(trace(enemy1,int3d[20,5,0],
  mobileenemy,red,0),trace(tank1,int3d[20,5,0],tank,
  red,0)),67,0
belief,marv1,new entity: trace(enemy1,int3d[20,5,0],
  mobileenemy,red,0),67,0
message,marv1,marg:traceinforadiomessage(trace(enemy1,
  int3d[20,5,0],mobileenemy,red,0)),67,0
belief,marg,new entity: trace(enemy1,int3d[20,5,0],
  mobileenemy,red,0),67,0
message,marg,command:traceinforadiomessage(trace(enemy1,
  int3d[20,5,0],mobileenemy,red,0)),67,0
belief,command,new entity: trace(enemy1,int3d[20,5,0],
  mobileenemy,red,0),67,0
belief,marv1,new entity: trace(tank1,int3d[20,5,0],tank,
  red,0),67,0
message,command,inf1: traceinforadiomessage(trace(enemy1,
  int3d[20,5,0],mobileenemy,red,0)),68,0
belief,inf1,new entity: trace(enemy1,int3d[20,5,0],
  mobileenemy,red,0),68,0
```

```
goal,inf1,be in fire range of target enemy1,68,0
plan,inf1,ground move to target enemy1,68,0
action,inf1,move towards (20,5,0),68,0
message,marv1,marg: traceinforadiomessage(trace(tank1,
  int3d[20,5,0],tank,red,0)),69,0
belief,marg,new entity: trace(tank1,int3d[20,5,0],tank,
  red,0),69,0
message,marg,command: traceinforadiomessage(trace(tank1,
  int3d[20,5,0],tank,red,0)),70,0
belief,command,new entity: trace(tank1,int3d[20,5,0],
  tank,red,0),70,0
message,command,inf1: traceinforadiomessage(trace(tank1,
  int3d[20,5,0],tank,red,0)),70,0
belief,inf1,new entity: trace(tank1,int3d[20,5,0],tank,red,0),
  70,0
goal,inf1,arrange cas for threat at int3d[20,5,0],70,0
plan,inf1,call for cas from command,70,0
message,inf1,command: casrequestmessage,71,0
goal,command,fulfil cas request,71,0
plan,command,task cas mission,71,0
message,command,ucav1: casmissionobjectivesmessage,71,0
goal,ucav1,air strike on int2d[20,5],71,0
plan,ucav1,fly to target,71,0
belief,ucav1, waypoint: int3d[20,5,1],71,0
belief,ucav1, waypoint: int3d[0,39,1],71,0
belief,marv1, arrived at waypoint int3d[22,12,1],72,0
belief,marv1, no more waypoints,72,0
belief,marv1, waypoint: int3d[0,0,1],72,0
message,marv1,marg: reconcompletemessage,73,0
message,marg,command: reconcompletemessage,73,0
message,command,inf1: reconcompletemessage,74,0
belief,ucav1, arrived at waypoint int3d[20,5,1],104,0
message,ucav1,command: casmissioncompletemessage,104,0
message,command,marg: bdarequestmessage,104,0
plan,marg,plan bda mission,104,0
action,ucav1,air strike at (20,5),104,0
event,tank1,destroyed by ucav1 at (20;5),104,0
message,marg,marv1: bdamissionordersmessage,105,0
plan,marv1,conduct bda,147,0
belief,marv1, arrived at waypoint int3d[20,5,1],190,0
belief,marv1, no more waypoints,190,0
```

```
message,marv1,marg: bdacompletemessage,192,0
message,marg,command: bdacompletemessage,193,0
belief,command,cas mission succeeded,193,0
message,command,inf1: cascompletemessage,193,0
belief,inf1,request cas complete,193,0
action,inf1,attack target enemy1,198,0
action,inf1,attack target enemy1,199,0
action,enemy1,attack target inf1,200,0
action,inf1,attack target enemy1,200,0
event,enemy1,destroyed by inf1 at (20;5),200,0
```

Glossary

AGO	Anti-Guerrilla Operations, one of the SoS used as a case study in this thesis
AS	Autonomous System
AWACS	Airborne Warning and Control System, an aircraft-based air-traffic control and surveillance platform operated by the US military
BDI	Belief-Desire-Intention, a model for describing the internal mechanisms of agents
COP	Combined Operational Picture, a computer-managed view of the world shared by all agents within a military force, maintained by means of networked data communications
CSO	Component System Owner, the owner of one of the systems that make up an SoS
DEVS	Discrete Event System Specification, a formalism for the expression of simulation models
DoDAF	Department of Defense Architecture Framework, an architecture description framework developed and promoted by the US Department of Defense
Domain Model	A model of a SoS expressed in terms comprehensible to domain experts
FFA	Function Failure Analysis, a hazard analysis technique similar to FHA
FHA	Functional Hazard Analysis, a hazard analysis technique that elicits hazards from a functional description of a system
FMEA	Failure Modes and Effects Analysis, a hazard analysis technique that derives possible hazardous effects from the known failure modes of system comments
GPS	Global Positioning System

HAZOP HAZard and OPerability study, a hazard analysis technique that guides analysts to derive hazards based on deviations of flows and processes

HLA High Level Architecture, a framework for simulation interoperability

IFF Identify Friend or Foe. An IFF system mounted on a vehicle can be remotely queried by another, and will respond with codes that identify the host vehicle type (e.g. military or civilian) and allegiance (e.g. USA)

ISTAR Intelligence, Surveillance, Target Acquisition and Reconnaissance

MANA Map-Aware Non-uniform Automata, a cellular automata model used for modelling military performance

MARG The ground station for the MARV

MARV A particular UAV used in the Adcock case study

MODAF Ministry of Defence Architecture Framework, the UK MOD's adaptation of DoDAF

NCW Network-Centric Warfare

OBEST Object-Based Event Scenario Tree, a method for exploring the possible paths taken by a non-deterministic simulation

Operational Model A model of a SoS expressed so as to be directly translatable into an imperative computer program

PHI Preliminary Hazard Identification, an initial brainstorming phase for the identification of hazards at a very early stage of system development

PSSA Preliminary System Safety Analysis, an assessment of the safety of a system in the early stages of development, based on a description of the system's architecture

QHI Qualitative Hazard Identifier, an automated HAZOP system

SA Situational Awareness, an agent's awareness of its surroundings, in terms of the location and disposition of itself, of other agents, and of terrain features

Scenario A description of a situation in which a SoS is involved, including mission goals, the geographical environment and the threats present

SEAS System Effectiveness Analysis Simulation, a simulation engine used for military performance modelling

SoS Accident An accident that occurs because of a SoS hazard

SoS Hazard	A condition of a SoS configuration, physical or otherwise, that can lead to an accident
SoS-HI	SoS Hazard Identification
SoS-PSSA	SoS Preliminary System Safety Analysis
SoS-RA	SoS Risk Assessment
Source Model	The model of an SoS originally received by the hazard analyst from the domain experts
STAMP	Systems Theoretic Accident Modelling and Processes
STPA	STamp Analysis, a hazard analysis technique based on the STAMP accident model
SysML	Systems Modelling Language, a notation for expressing systems engineering models
TOPAZ	Traffic Organisation and Perturbation AnalyZer, a method for assessing safety risk in air traffic systems
TPL	Tactical Programming Language, a domain-specific language used to script agent behaviours in the SEAS simulation engine
UAV	Unmanned Air Vehicle
UCAV	Unmanned Combat Air Vehicle
UGV	Unmanned Ground Vehicle
UML	Unified Modelling Language, a notation for expressing software architectures and designs
VID	Visual IDentification, an attempt to determine the identity of an platform based on its visual characteristics
Vignette	A fraction of a scenario that is small enough in terms of time and geographic extent to be rendered entirely within a simulation engine

List of References

[1] A. Pyster and P. Gardner, "Critical success factors in system of systems engineering," SAIC Corporation, October 2006, http://sunset.usc.edu/events/2006/CSSE_Convocation/presentations/Pyster.ppt, accessed 20 Jan 2007.

[2] D. Pumfrey, "The principled design of computer system safety analysis," DPhil Thesis, University of York, 2000.

[3] D. S. Caffall and J. B. Michael, "System-of-systems design from an object-oriented paradigm," in *Proceedings of the Monterey Workshop: Radical Innovations of Software and Systems Engineering in the Future*. Venice, Italy: U. S. Army Research Office, 2002, pp. 146–157.

[4] M. Hall-May and T. P. Kelly, "Using agent-based modelling approaches to support the development of safety policy for systems of systems," in *Proceedings of the 25th International Conference on Computer Safety, Reliability and Security (SAFECOMP '06)*, ser. LNCS, J. Gorski, Ed., vol. 4166. Gdansk, Poland: Springer-Verlag, Sep. 2006, pp. 330–343.

[5] A. Drogoul, D. Vanbergue, and T. Meurisse, "Multi-agent based simulation: Where are the agents ?" in *Multi-Agent-Based Simulation II: Third International Workshop, MABS 2002, Bologna, Italy, July 15-16, 2002. Revised Papers*, 2002.

[6] L. Padgham and M. Winnikoff, *Developing Intelligent Agent Systems: a Practical Guide*. John Wiley & Sons, 2004.

[7] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson, 2002.

[8] A. Villemeur, *Reliability, Availability, Maintainability and Safety Assessment: Volume 1 — Methods and Techniques*. Chicester, England: John Wiley & Sons, 1992.

[9] C. W. Johnson, *Failure in Safety-Critical Systems: a Handbook of Accident and Incident Reporting*. Glasgow: University of Glasgow Press, 2003.

[10] NECTISE, "Issue 1 — operational architecture," Unpublished technical report.

[11] "MIL-STD-882D — standard practice for system safety," Department of Defense, Feb 2000.

[12] D. N. Lam and K. S. Barber, "Verifying and explaining agent behaviour in an implemented agent system," Laboratory for Intelligent Processes and Systems, Department of Electrical and Computer Engineering, The University of Texas at Austin, Tech. Rep. TR2004-UT-LIPS-000, 2004.

[13] N. Leveson, P. Allen, and M.-A. Storey, "The analysis of a friendly fire accident using a systems model of accidents," in *Proceedings of the 20th International System Safety Society Conference (ISSC 2002)*. System Safety Society, Unionville, Virginia, 2002, pp. 345–357.

[14] S. A. Snook, *Friendly Fire: the Accidental Shootdown of U.S. Black Hawks Over Northern Iraq*. Princeton, New Jersey: Princeton University Press, 2000.

[15] G. Regan, *Backfire*. Robson Books, 2002.

[16] D. Craig, D. Morales, and M. Oliver, "USS Vincennes incident," MIT Aeronautics & Astronautics, 2004.

[17] M. Burgess, "Killing your own: the problem of friendly fire during the afghan campaign," Center for Defence Information, June 2002, http://www.cdi.org/terrorism/killing.cfm, accessed 21 April 2007.

[18] J. C. Knight and S. M. Parikh, "Simulation technology for free flight system performance and survivability analysis," in *Proceedings of the 21st Digital Avionics Systems Conference*, Irvine, CA, November 2002.

[19] *JSP 777: Network Enabled Capability*, UK Ministry of Defence, November 2004.

[20] "ASAAC standards — part 1 — proposed standards for software," MoD Interim Defence Standard 00-74 Issue 1, January 2005.

[21] C. Perrow, *Normal Accidents: Living with High-Risk Technologies*. New York: Basic Books, 1984.

[22] N. Leveson, "A new accident model for engineering safer systems," in *Proceedings of the 20th International System Safety Society Conference (ISSC 2003)*. System Safety Society, Unionville, Virginia, 2002, pp. 476–486.

[23] O. Lisagor, J. A. McDermid, and D. J. Pumfrey, "Towards a practicable process for automated safety analysis," in *Proceedings of the 24th International System Safety Conference (ISSC)*. Albuquerque, NM: Systems Safety Society, August 2006.

[24] *Defence Acquisition Guidebook*, 1st ed., US Department of Defence, July 2006.

[25] M. W. Maier, "Architecting principles for systems-of-systems," in *6th Annual Symposium of INCOSE*, 1996, pp. 567–574.

[26] A. W. A. Owens, "The emerging systems of systems," *Naval Institute Proceedings*, vol. 121, pp. 35–39, 1995.

[27] V. Kotov, "Systems of systems as communicating structures," Hewlett-Packard Computer Systems Laboratory, Tech. Rep., 1997.

[28] Dependable Systems of Systems Project, "Dependable systems of systems project summary," June 2003, http://www.newcastle.research.ec.org/dsos/programme/summary.html.

[29] M.-C. Gaudel, V. Issarny, C. Jones, H. Kopetz, E. Marsden, N. Moffat, M. Paulitsch, D. Powell, B. Randell, A. Romanovsky, R. Stroud, and F. Taiani, "Final version of DSOS conceptual model (CSDA1)," University of Newcastle upon Tyne, Tech. Rep. CS-TR-782, 2003.

[30] P. Periorellis and J. Dobson, "Organisational failures in dependable collaborative enterprise systems," *Journal of Object Technology*, vol. 1, no. 3, pp. 107–117, 2002.

[31] D. Alberts, J. Garstka, and F. Stein, *Network Centric Warfare: Developing and Leveraging Information Superiority*. CCRP Publications, 1999.

[32] J. M. House, *Combined Arms Warfare in the Twentieth Century*. Lawrence, Kansas: University Press of Kansas, 2001.

[33] C. Keating, R. Rogers, R. Unal, D. Dryer, A. Sousa-Poza, R. Safford, W. Peterson, and G. Rabadi, "System of systems engineering," *Engineering Management Journal*, vol. 15, no. 3, pp. 36–45, September 2003.

[34] P. Checkland, *Systems Thinking, Systems Practice*. John Wiley & Sons, 1999.

[35] D. Raheja and B. Moriarty, "New paradigms in system safety," *Journal of System Safety*, vol. 42, no. 6, Nov–Dec 2006.

[36] S. Johnson, *Emergence: the Collected Lives of Ants, Brains, Cities and Software*. Penguin Press, 2001.

[37] K. Kelly, *Out of Control: the new biology of machines, social systems and the economic world*. Reading, Mass.: Addison-Wesley, 1994.

[38] M. Resnick, "Beyond the centralized mindset," *Journal of the Learning Sciences*, vol. 5, no. 1, pp. 1–22, 1996.

[39] Dependable Systems of Systems Project, "Annex 1 - description of work," January 2000, http://www.newcastle.research.ec.org/dsos/programme/dow_public.pdf.

[40] N. Leveson and N. Dulac, "Safety and risk driven design in complex systems of systems," in *Proceedings of the 1st NASA/AIAA Space Exploration Conference*, Orlando, Florida, February 2005.

[41] C. W. Johnson and C. M. Holloway, "The ESA/NASA SOHO mission interruption: Using the STAMP accident analysis technique for a software related 'mishap'," *Software: Practice and Experience*, vol. 33, pp. 1177–1198, 2003.

[42] J. McDermid, Ed., *Software Engineer's Reference Book*. Butterworth-Heinemann, 1991.

[43] System Safety Society, *System Safety Analysis Handbook*. System Safety Society, 1997.

[44] P. Fenelon, J. A. McDermid, M. Nicolson, and D. J. Pumfrey, "Towards integrated safety analysis and design," *ACM SIGAPP Applied Computing Review*, vol. 2, no. 1, pp. 21–32, March 1994.

[45] SAE, "Aerospace recommended practice 4761 — guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment," Society of Automotive Engineers, Tech. Rep., 1996.

[46] CISHEC, *A Guide to Hazard and Operability Studies*. The Chemical Industry Safety and Health Council of the Chemical Industries Association Ltd, 1977.

[47] N. Leveson, "A new approach to hazard analysis for complex systems," in *Proceedings of the 21st International System Safety Conference (ISSC)*. Ottawa, Canada: Systems Safety Society, August 2003.

[48] V. Venkatasubramanian, J. Zhao, and S. Viswanathan, "Intelligent systems for HAZOP analysis of complex process plants," *Computers and Chemical Engineering*, vol. 24, pp. 2291–2302, 2000.

[49] C. A. Catino and L. H. Ungar, "Model-based approach to automated hazard identification of chemical plants," *AIChE Journal*, vol. 41, no. 1, pp. 97–109, 1995.

[50] B. Kuipers, *In R. A. Meyers (Ed.), Encyclopedia of Physical Science and Technology*, 3rd ed. NY: Academic Press, 2001, ch. Qualitative simulation.

[51] Y. Papadopoulos, D. Parker, and C. Grante, "A method and tool support for model-based semi-automated failure modes and effects analysis of engineering designs," in *Proceedings of the 9th Australian Workship on Safety Critical Systems (SCS 2004)*, 2004.

[52] M. Wallace, "Modular architectural representation and analysis of fault propagation and transformation," in *Proceedings of the Second International Workshop on Formal Foundations of Embedded Software and Component-based Software Architectures (FESCA 2005)*, ser. Electronic Notes in Theoretical Computer Science, vol. 141, no. 3. Elsevier B.V., December 2005, pp. 53–71.

[53] J. Voas, "Error propagation analysis for COTS systems," *Computing & Control Engineering Journal*, 1997.

[54] H. A. P. Blom, S. H. Stroeve, and H. H. de Jong, "Safety risk assessment by Monte Carlo simulation of complex safety critical operations," in *Proceedings of the Fourteenth Safety-critical Systems Symposium*, F. Redmill and T. Anderson, Eds., Safety-Critical Systems Club. Bristol, UK: Springer, 2006, pp. 47–67.

[55] H. A. P. Blom, G. J. Bakker, P. J. G. Blanker, J. Daams, M. H. C. Everdij, and M. B. Klompstra, *Air Transportation Systems Engineering*. American Institute of Aeronautics and Astronautics, 2001, vol. 193, ch. Accident Risk Assessment for Advanced Air Traffic Management, pp. 463–480.

[56] J. A. Dewar, S. C. Bankes, J. S. Hodges, T. Lucas, D. K. Saunders-Newton, and P. Vye, "Credible uses of the distributed interactive simulation (DIS) system," RAND, Tech. Rep. MR-607-A, 1996.

[57] M. K. Lauren and R. T. Stephen, "Map-Aware Non-Uniform Automata (MANA): A New Zealand approach to scenario modelling," *Journal of Battlefield Technology*, vol. 5, no. 1, pp. 27–31, March 2002.

[58] D. P. Galligan, "Modelling shared situational awareness using the MANA model," *Journal of Battlefield Technology*, vol. 7, no. 3, pp. 35–40, November 2004.

[59] H. Brooks, T. DeKeyser, D. Jaskot, D. Sibert, R. Sledd, W. Stilwell, and W. Scherer, "Using event-based simulation to reduce collateral damage during military operations," in *Proceedings of the 2004 Systems and Information Engineering Design Symposium*, M. H. Jones, S. D. Patek, and B. E. Tawney, Eds., 2004, pp. 71–78.

[60] C. Johnson, "The Glasgow-hospital evacuation simulator: Using computer simulations to support a risk-based approach to hospital evacuation," University of Glasgow, Tech. Rep., 2005, submitted to the Journal of Risk and Reliability.

[61] N. Leveson, "Advanced system and safety engineering environments," http://sunnyday.mit.edu/spectrm.ppt.

[62] German Federal Bureau of Aircraft Accidents Investigation, "Investigation report AX001-1-2/02," 2004.

[63] Department of Defence Architecture Framework Working Group, "DoD architecture framework v. 1.0 — volume I: Definitions and guidelines," DoD, February 2004.

[64] MODAF Partners, "MOD architectural framework executive summary," Ministry of Defence, August 2005, version 1.0.

[65] S. Mittal, "Extending DoDAF to allow integrated DEVS-based modeling and simulation," *Journal of Defense Modeling and Simulation*, vol. 3, no. 2, pp. 95–123, April 2006.

[66] Department of Defence Architecture Framework Working Group, "DoD architecture framework v. 1.0 — volume II: Product descriptions," DoD, February 2004.

[67] A. W. Zinn, "The use of integrated architectures to support agent based simulation: an initial investigation," Master's thesis, Air Force Institute of Technology, March 2004.

[68] A. Ilachinski, "Exploring self-organized emergence in an agent-based synthetic warfare lab," *Kybernetes: The International Journal of Systems & Cybernetics*, vol. 32, no. 1, pp. 38–76, February 2003 2003.

[69] F. W. Lanchester, "Aircraft in warfare," *Engineering*, vol. 98, pp. 422–423, 1914, (Reprinted on pages 2138–2148, The World of Mathematics, Volume IV, edited by J. Newman, Simon and Schuster, 1956.).

[70] B. P. Zeigler, H. Praehofer, and T. G. Kim, *Theory of Modeling and Simulation. Integrating Discrete Event and Continuous Complex Dynamic Systems*, 2nd ed. Academic Press, 2000.

[71] MODAF Partners, "MODAF handbook volume 2: Technical specification for modaf," www.modaf.org.

[72] R. Alexander, M. Hall-May, and T. Kelly, "Characterisation of systems of systems failures," in *Proceedings of the 22nd International Systems Safety Conference (ISSC 2004)*. System Safety Society, 2004, pp. 499–508.

[73] B. T. Clough, "Metrics, schmetrics! how the heck do you determine a UAV's autonomy anyway?" in *Proceddings of the 2002 PerMis Workshop*. Gaithersburg, MD: NIST, August 2002, pp. 1–7.

[74] C. W. Johnson and C. M. Holloway, "On the over-emphasis of human 'error' as a cause of aviation accidents: 'systemic failures' and 'human error' in US NTSB and Canadian TSB aviation reports 1996–2003," in *Proceedings of the 22nd International System Safety Conference (ISSC)*. Providence, RI: Systems Safety Society, August 2004.

[75] J. McDermid, personal communication, July 2007.

[76] D. A. Norman, "The 'problem' with automation: Inappropriate feedback and interaction, not 'over-automation'," *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, vol. 327, no. 1241, pp. 584–593, April 1990.

[77] R. G. Sargent, R. E. Nance, C. M. Overstreet, S. Robinson, and J. Talbot, "The simulation project life-cycle: models and realities," in *Proceedings of the 37th Winter Simulation Conference*, 2006, pp. 863–871.

[78] T. Kletz, *HAZOP and HAZAN: Identifying and Assessing Process Industry Hazards*, 3rd ed. Institution of Chemical Engineers, 1992.

[79] J. Ferber, *Multi-Agent Systems: an Introduction to Distributed Artificial Intelligence*. Addison-Wesley, 1999.

[80] R. C. Zittel, "The reality of simulation-based acquisition — and an example of US military implementation," *Acquisition Review Quarterly*, 2001.

[81] M. Hall-May and T. P. Kelly, "Defining and decomposing safety policy for systems of systems," in *Proceedings of the 24th International Conference on Computer Safety, Reliability and Security (SAFECOMP '05)*, ser. LNCS, R. Wither, B. A. Gran, and G. Dahll, Eds., vol. 3688. Fredrikstad, Norway: Springer-Verlag, Sep. 2005, pp. 37–51.

[82] G. Mitchell, *The Practice of Operational Research*. John Wiley & Sons, 1993.

[83] Object Management Group, "Unified modelling language: Superstructure," February 2005, http://www.omg.org/cgi-bin/apps/doc?formal/07-02-05.pdf.

[84] SML Partners, "Systems modeling language (SysML) specification," sysml.org, May 2006, version 1.0, http://www.sysml.org/docs/specs/OMGSysML-FAS-06-05-04.pdf.

[85] T. Mahmood and E. Kazmierczak, "A knowledge-based approach for safety analysis using system interactions," in *Proceedings of the Asia Pacific Software Engineering Conference, APSEC'06*. IEEE Computer Society, December 2006.

[86] G. Weinberg, *An Introduction to General Systems Thinking*, silver anniversary ed. Dorset House Publishing Company, 2001.

[87] F. P. Hoeber, *Military Applications of Modeling: Selected Case Studies*. Gordon & Breach Science Publishers, 1981.

[88] R. G. Sargent, "Verification, validation, and accreditation of simulation models," in *Proceedings of the 2000 Winter Simulation Conference*, 2000.

[89] G. Despotou, "DODAF model development methodology; the ago example," University of York, Tech. Rep. DARP/TR/2005/16, 2005.

[90] Columbia Accident Investigation Board, "Report volume 1," National Aeronautics and Space Administration, August 2003.

[91] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard, *Object-Oriented Software Engineering: A Use-Case Driven Approach*. Reading, MA: Addison-Wesley, 1992.

[92] Defence Modelling and Simulation Office, "Online M&S glossary (DoD 5000.59-m)," Department of Defence, October 2005, https://www.dmso.mil/public/resources/glossary/.

[93] M. K. Nicole Ronald, Leon Sterling, "An agent-based approach to modelling pedestrian behaviour," *International Journal of Simulation*, vol. 8, no. 1, pp. 25–38, February 2007.

[94] M. Bratman, *Intention, Plans, and Practical Reason*. Cambridge, MA: Harvard University Press, 1987.

[95] D. McIlroy and C. Heinze, "Air combat tactics implementation in the smart whole air mission model (SWARMM)," in *Proceedings of the First International SimTecT Conference*, Melbourne, Australia, 1996.

[96] J. Sudeikat, L. Braubach, A. Pokahr, and W. Lamersdorf, "Evaluation of agent-oriented software methodologies — examination of the gap between modeling and platform," in *Proceedings of Agent-Oriented Software Engineering V (AOSE 2004)*. New York, NY, USA: Springer Berlin / Heidelberg, 2005, pp. 126–141, lNCS Volume 3382.

[97] L. Padgham, J. Thangarajah, and M. Winikoff, "Tool support for agent development using the Prometheus methodology," in *Proceedings of the first international workshop on Integration of Software Engineering and Agent Technology (ISEAT 2005)*, Melbourne, Australia, 2005.

[98] C. Larman, *Applying UML and Patterns*. Prentice Hall PTR, 2002.

[99] L. Padgham and M. Winikoff, "Prometheus: A methodology for developing intelligent agents," in *Proceedings of the Third International Workshop on Agent-Oriented Software Engineering*, 2002.

[100] A. M. Law, *Simulation Modeling and Analysis*, 4th ed. McGraw-Hill, 2007.

[101] J. Suokas and R. Kakko, "On the problems and future of safety and risk analysis," *Journal of Hazardous Materials*, vol. 21, pp. 105–124, 1989.

[102] R. Hawkins, I. Toyn, and I. Bate, "An approach to designing safety critical systems using the Unified Modelling Language," in *Critical Systems Development with UML — Proceedings of the UML'03 Workshop*, J. Jürjens, B. Rumpe, R. France, and E. B. Fernandez, Eds. Technische Universität München, September 2003, pp. 3–17.

[103] J. Górski and B. Nowicki, "Object-oriented approach to safety analysis," in *Proceedings of the 12th Annual CSR Workshop*, 1995, pp. 338–350.

[104] D. J. Smith, "Developments in the use of failure rate data and reliability prediction methods," Ph.D. dissertation, Delft University of Technology, 2000.

[105] J. C. Williams, "HEART - a proposed method for assessing and reducing human error," in *Proceedings of the 9th Advances in Reliability Technology Symposium*, University of Bradford, 1986.

[106] P. B. Ladkin, "Taking software seriously," *Journal of System Safety*, vol. 41, no. 3, May–June 2005.

[107] B. Littlewood and L. Strigini, "Assessment of ultra-high dependability of software-based systems," *Communications of the ACM*, vol. 36, no. 11, pp. 69–80, November 1993.

[108] IEC, "IEC 61508, functional safety of electrical/electronic/programmable electronic safety-related systems," January 2005.

[109] P. Bishop, "SILs and software," Adelard and Centre for Software Reliability, City University, http://www.adelard.com/papers/SCSC_Newsletter_Software_SILs.pdf.

[110] F. Redmill, "Understanding the use, misuse and abuse of safety integrity levels," Redmill Consulting, February 2000, http://www.csr.ncl.ac.uk/FELIX_Web/3A.SILs.pdf, accessed 17 April 2007.

[111] V. P. Brand, *UPM 3.1: A pragmatic approach to dependent failures assessment for standard systems*, AEA Technology, April 1996.

[112] "Safety management requirements for defence systems," MoD Interim Defence Standard 00-56 issue 3, December 2004.

[113] M. Ha-Duong, "Scenarios, probability and possible futures," April 2006, http://minh.haduong.com/files/HaDuong-2006-ScenariosProbabilityPossibleFutures.pdf.

[114] R. Hastie and R. M. Dawes, *Rational Choice in an Uncertain World*. SAGE Publications Ltd, 2001.

[115] N. Howden, R. Rönnquist, A. Hodgson, and A. Lucas, "Jack — summary of an agent infrastructure," in *Proceedings of the 5th International Conference on Autonomous Agents*, 2001.

[116] A. Pokahr, L. Braubach, and W. Lamersdorf, *Multi-Agent Programming*. Kluwer, 2005, ch. Jadex: A BDI Reasoning Engine.

[117] G. J. Humphreys-Jones, "Force level analysis and mission effectiveness simulation (FLAMES)," in *IEE Colloquium on Computer Modelling and Simulation of Radar Systems*, London, UK, February 1993, pp. 3/1–3//4.

[118] R. L. Wittman Jr and C. T. Harrison, "OneSAF: A product line approach to simulation development," in *Proceedings of the European Simulation Interoperability Workshop*, June 2001.

[119] NECTISE, "Issue 2 — capability context," Unpublished technical report.

[120] R. Alexander, "A prototype simulation engine for hazard analysis," University of York, Tech. Rep. DARP/TR/2005/12, 2005.

[121] S. Luke, C. Cioffi-Revilla, L. Panait, and K. Sullivan, "MASON: A new multi-agent simulation toolkit," in *Proceedings of the 2004 SwarmFest Workshop*, 2004.

[122] Health and Safety Executive, "Out of contol: Why control systems go wrong and how to prevent failure," 1995.

[123] R. H. Bordini and J. F. Hübner, "BDI agent programming in AgentSpeak using Jason," in *Proceedings of CLIMA VI*, 2005, pp. 142–164.

[124] D. C. Montgomery, *Design and Analysis of Experiments*, 5th ed. John Wiley & Sons, 2001.

[125] P. Gray, W. Hart, L. Painton, C. P. M. Trahan, and J. Wagner, "A survey of global optimization methods," Sandia National Laboratories, 1997, http://www.cs.sandia.gov/opt/survey/.

[126] K. A. Richardson, "'methodological implications of complex systems approaches to sociality': Some further remarks," *Journal of Artificial Societies and Social Simulation*, vol. 5, no. 2, 2002.

[127] B. Edmonds, "The use of models — making MABS more informative," in *Proceedings of the 3rd International Workship on Multi-Agent Systems and Agent-Based Simulation (MABS 2000)*, S. Moss and P. Davidson, Eds. Springer-Verlag Berlin Heidelberg, 2000, pp. 15–32.

[128] C. Zhao and V. Venkatasubramanian, "Learning in intelligent systems for process safety analysis," in *Proceedings of ESCAPE-15*, 2005.

[129] J. Fox and S. Das, *Safe and Sound: Artificial Intelligence in Hazardous Applications*. MIT Press, 2000.

[130] F. Ammirato, M. Bieth, O. J. V. Chapman, L. M. Davies, G. Engl, C. Faidy, T. Seldis, D. Szabo, P. Trampus, K.-S. Kang, and J. Zdarek, "Improvement of in-service inspection in nuclear power plants," International Atomic Energy Agency, Tech. Rep. IAEA-TECDOC-1400, 2004.

[131] National Institue of Standards and Technology, "Bin packing problem," 2004, http://www.nist.gov/dads/HTML/binpacking.html.

[132] National Institute of Standards and Technology, "Knapsack problem," 2005, http://www.nist.gov/dads/HTML/knapsackProblem.html.

[133] Å. Svensson, "A method for analysing the traffic process in a safety perspective," Dept. of Traffic Planning and Engineering, Lund University, Tech. Rep., 1998.

[134] C. Mellish and W. Clocksin, *Programming in PROLOG*, 4th ed. Springer-Verlag Berlin and Heidelberg GmbH & Co. K, 1994.

[135] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[136] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*, 2nd ed. San Francisco: Morgan Kaufmann, 2005.

[137] R. A. Fisher, *Contributions to Mathematical Statistics*. John Wiley, 1950, ch. The use of multiple measurements in taxonomic problems.

[138] P. D. Picton, *Neural Networks*, 2nd ed. Palgrave, 2000.

[139] G. Towell and J. Shavlik, "Knowledge-based artificial neural networks," *Connection Science*, vol. 1, pp. 233–255, 1989.

[140] Z. Kurd, "Artificial neural networks in safety critical applications," Ph.D. dissertation, Department of Computer Science, University of York, 2006.

[141] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[142] S. H. Muggleton, Ed., *Inductive Logic Programming*. London: Academic Press, 1992.

[143] D. Page and A. Srinivasan, "ILP: A short look back and a longer look forward," *Journal of Machine Learning Research*, vol. 4, pp. 415–430, 2003.

[144] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kauffman, 1993.

[145] J. R. Quinlan, "Simplifying decision trees," *International Journal of Man Machine Studies*, vol. 27, pp. 221–234, 1987.

[146] D. Michie and I. Bratko, *Expert Systems — Automating Knowledge Acquisition*. Addison-Wesley Publishing Company, 1986.

[147] J. Darlington, Y. ke Guo, J. Sutiwaraphun, and H. W. To, "Parallel induction algorithms for data mining," in *Advances in Intelligent Data Analysis. Reasoning about Data: Second International Symposium, IDA-97*, London, UK, 1997, pp. 437–446.

[148] P. Becuzzi, M. Coppola, S. Ruggieri, and M. Vanneschi, "Parallelisation of C4.5 as a Particular Divide and Conquer Computation," in *Parallel and Distributed Processing*, ser. Lecture Notes in Computer Science, J. Rolim *et al.*, Eds., vol. 1800. Cancun, Mexico: Springer, May 2000, pp. 382–389, 3rd Workshop on High Performance Data Mining, IPDPS 2000.

[149] N. V. Chawla, "C4.5 and imbalanced data sets: Investigating the effect of sampling method, probabilistic estimate, and decision tree structure," in *Proceedings of the Workshop on Learning from Imbalanced Datasets II*, Washington, DC, 2003.

[150] "MIL-STD-882E — standard practice for system safety (draft)," Department of Defense, Feb 2006.

[151] J. Abrams and R. Cone, "Implementing expected monetary value analysis into risk metrics and assessment criteria," in *Proceedings of the 24th International System Safety Conference (ISSC)*. Albuquerque, NM: Systems Safety Society, August 2006.

[152] R. Ekholm, "Summation — from individual risks to total system risk," in *Proceedings of the 24th International System Safety Conference (ISSC)*. Albuquerque, NM: Systems Safety Society, August 2006.

[153] G. D. Wyss, F. A. D. Felicia, and V. J. Dandini, "An object-oriented approach to risk and reliability analysis: methodology and aviation safety applications," *Simulation*, vol. 80, no. 1, pp. 33–43, 2004.

[154] D. N. Lam and K. S. Barber, "Comprehending agent software," in *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2005)*, 2005.

[155] T. Bosse, D. N. Lam, and K. S. Barber, "Automated analysis and verification of agent behavior," in *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS '06)*, P. Stone and G. Weiss, Eds. ACM Press, 2006.

[156] P. Humphreys, *Extending Ourselves - Computational Science, Empiricism, and Scientific Method.* New York: Oxford University Press, 2004.

[157] K. Richardson, G. Mathieson, and P. Cilliers, "The theory and practice of complexity science — epistemological considerations for military operational analysis," *SysteMexico*, vol. 1, pp. 25–66, 2000.

[158] J. C. Hill, "Resolving complexity in accident texts through graphical notations and hypertext," DPhil Thesis, University of York, 2001.

[159] T. P. Kelly, "Arguing safety—a systematic approach to managing safety cases," DPhil Thesis, University of York, Heslington, York, YO10 5DD, UK, Sep. 1998.

[160] J. S. Dahmann, R. M. Fujimoto, and R. M. Weatherly, "The DoD high level architecture: An update," in *Proceedings of the 1998 Winter Simulation Conference*, Washington, DC, 1998, pp. 797–804.

[161] T. I. Oren, S. K. Numrich, A. M. Uhrmacher, L. F. Wilson, and E. Gelenbe, "Agent-directed simulation - challenges to meet defense and civilian requirements," in *Proceedings of the 2000 Winter Simulation Conference*, vol. 2, 2000, pp. 1757–1762.

[162] Y. Mao, J. Vassileva, and W. Grassmann, "A system dynamics approach to study virtual communities," in *Proceedings of the HICSS'07 mini-track on Virtual Communities*, Big Island, Hawaii, January 2007.

[163] H. V. D. Parunak, R. Savit, and R. L. Riolo, "Agent-based modeling vs. equation-based modeling: A case study and users' guide," in *Proceedings of the 1st International Workship on Multi-Agent Systems and Agent-Based Simulation (MABS '98)*, J. S. Sichman, R. Conte, and N. Gilbert, Eds., Paris, France, July 1998, pp. 10–25.

[164] J. F. Dunnigan, *Wargames Handbook: How to Play and Design Commercial and Professional Wargames*, 3rd ed. iUniverse.com, 2000.

[165] E. Norling, "Folk psychology for human modelling: Extending the BDI paradigm," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, New York, NY, July 2004.

[166] T. W. Lucas, "The stochastic versus deterministic argument for combat simulations: Tales of when the average won't do," *Military Operations Research*, vol. 5, no. 3, 2000.

[167] D. A. Cohn, Z. Ghahramani, and M. I. Jordan, "Active learning with statistical models," *Journal of Artificial Intelligence Research*, vol. 4, pp. 129–145, 1996.

[168] P. B. Ladkin, "A quick introduction to Why-Because analysis," 1999. [Online]. Available: http://www.rvs.uni-bielefeld.de/research/WBA/introWB.ps