ARCHITECTURE-BASED SELECTION OF MODELING TYPE FOR SYSTEM OF SYSTEMS ANALYSIS

A Dissertation
Presented to
The Academic Faculty

by

Burak Bağdatlı

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology
December, 2018

ARCHITECTURE-BASED SELECTION OF MODELING TYPE FOR SYSTEM OF SYSTEMS ANALYSIS

Approved by:

Professor Dimitri N. Mavris, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Professor Daniel P. Schrage
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Kelly Griendling
Center of Innovation for Aerospace
Georgia Department of Economic Development

Dr. Benjamin Poole
Space, Missile, and Nuclear Systems
MITRE Corporation

Dr. Olivia Fischer
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: August 1, 2018

To the women

who brought me up and made me the man I am today.

ACKNOWLEDGEMENTS

It is said that a thesis research is a very personal journey, but this work would not have been possible without the support I have received from a good number of people who were close to me or gotten close to me while giving me directions or walking a similar path.

First of all, I would like to thank my thesis advisor Prof. Dimitri Mavris. Every time I was in trouble, stuck, or out of ideas, I ran to him for help. His systematic ways of thinking and seemingly endless knowledge about every random technical subject helped me greatly to see the big picture while keeping me honest about the details. An advisor has a significant impact on a student and that impact lasts a lifetime. I have never met anyone who works as hard as him and I hope to be worthy of his coaching. He is like a second father to me and I will never forget his support during these intellectually and emotionally stressful years.

My thesis committee's amount of effort on reviewing my work and helping me with feedback has been incredibly impactful. Dr. Kelly Griendling, who was with me since I was preparing thesis ideas to pitch them to Prof. Mavris, has been hugely helpful. I hope I have not tired her out in this long journey. She always believed me even when I did not see any reason to believe myself. Prof. Daniel Schrage, who have been on this journey with me since my proposal, who always listened to my ideas and research attentively, and suggested opportunities along the way that I was not able to see. His enthusiasm and experience always had me leave his office in an elated and hopeful mood. Dr. Olivia Fischer, whose work ethic and stamina is unparalleled has been a great example for me to follow, has spent hours after hours, days after days, on my document. I enjoyed working with her in other research areas

as well. And Dr. Ben Poole, whom I met during an external advisory board review and who was always interested and attentive to my research ideas and work. I did not have the opportunity to work with him in person, but our communications were very productive and his technical knowledge in the field has made my work significantly more relevant and grounded in practical realities.

I have always believed that education is a social process. All my friends at Aerospace Systems Design Laboratory helped me in some way throughout this journey whether they know it or not. Certainly, my classmates in my first year have all been examples of what a good student is. Looking back now, our first troubles and difficulties in that first year seem minor problems, but they helped us forge a friendship that lasted long years, and those challenges seem easy now because we have all grown and each developed intellectual superpowers. I will always be thankful for their acceptance of my quirks, cultural background, and all the awkwardness they had to endure occasionally. I especially would like to mention Dr. Annie Jones Wyatt who has been such a good friend to me and whose thesis I used in part of my work. Also a special thanks to my partner in crime Dr. Mike Miller who walked not just the thesis journey but also the teaching assistantship journey with me. We spent countless hours toiling over exams, reports, and class preparations together and I certainly appreciated his help in every step of the way. I hope I was as helpful to him as he was to me. Metin Özcan, Dr. Nick Molino, Dr. Andy Turner, Dr. Charles Potter, Dr. Dan Garmendia, Dr. Brad Robertson, Dr. Imon Chakraborty, Dr. Derya Aksaray, Dr. David Jackson, Dr. Jonathan Sharma and countless others have been in my support community for a long time. I hope I was as good of a friend and support to them as they were to me.

I am also hugely thankful for the continued contact and support from my high school friends. Living in a foreign country has its difficulties and I was always able to chat, call, game with my old friends even when I was many thousands of miles

away from them. I am thankful that whenever I go back, they still like me and treat me like a close friend. I think the bonds that we forged in middle school and high school are stronger than Einstein's spacetime and cannot be broken by miles or years that are in between us. A special thanks to İlker, Ömer, Onur, Aras, Alican, Arda, Römer, Gamze, and countless others for never allowing me to feel alone when I was alone.

And finally, my family, my greatest support throughout. I left you when I decided to study aerospace engineering for reasons that can only be described teenage whims. I do not know why you did not try to stop me but instead supported me throughout. I have missed many happy and sad moments with you while I was away and I am deeply sorry for not being there to share those moments with you. I hope you do not regret sending me away. My mother, father, and brother who always greeted me with seemingly infinite love when I *visited home*. I love you all and hope to repay all the sacrifices you have made for me one day. Anne, baba, Barlas, hepinizi öpüyorum.

TABLE OF CONTENTS

DEDICATION . . . . . **iii**

ACKNOWLEDGEMENTS . . . . . **iv**

LIST OF TABLES . . . . . **xvii**

LIST OF FIGURES . . . . . **xxv**

LIST OF ACRONYMS AND ABBREVIATIONS . . . . . **xxxiii**

GLOSSARY . . . . . **xxxv**

LIST OF SYMBOLS . . . . . **xxxvi**

SUMMARY . . . . . **xxxix**

I INTRODUCTION **1**

1.1 Motivating problem 5

II ON THE DESIGN OF SYSTEMS **10**

2.1 Engineering design 11

2.2 Systems 16

2.3 Systems of systems 19

2.4 Architectures 25

2.5 Modeling and simulation 32

III RESEARCH ARGUMENTS AND WORK **36**

3.1 Gaining knowledge via synthetic arguments 36

3.2 Motivating characteristics of the problem 38

3.3 Induction: Architectures as conceptual model 45

3.3.1 Experiment 1 47

3.4 Hypothesis: A multitude of modeling techniques are needed 50

3.4.1 Experiment 2 54

3.5 Using the right models for the right problem 56

IV ON MODELS 59
4.1 Definitions of a model 60
4.2 Scientific models 63
4.3 Engineering models 72
4.3.1 Non-existence of the real system 73
4.3.2 Replacement of the real system 74
4.3.3 Providing accurate observations 77
4.4 Computer models 80
4.4.1 Verification and validation of computer models 83
4.5 Working definition of a model 88
V SYSTEM OF SYSTEMS MODELING 93
5.1 Design description 95
5.2 Models for system of systems 98
5.3 Deterministic and static models 103
5.3.1 Graph theory 103
5.3.2 Probabilistic calculations 106
5.4 Deterministic, dynamic, and continuous models 112
5.4.1 Continuous-time Markov chains 112
5.4.2 System dynamics 118
5.5 Deterministic, dynamic, and discrete models 122
5.5.1 Discrete-time Markov chains 122
5.5.2 Petri nets 126
5.6 Stochastic and static models 128
5.6.1 Monte Carlo 128
5.7 Stochastic, dynamic, and continuous models 129
5.8 Stochastic, dynamic, and discrete models 130
5.8.1 Monte Carlo discrete time Markov chains 130
5.8.2 Petri nets 131

5.8.3 Queueing theory . . . 132
5.8.4 Discrete event . . . 134
5.8.5 Agent-based . . . 137
5.9 Transition to experiments and technical work . . . 139

VI MODELING POTENTIAL OF OPERATIONAL ARCHITECTURE VIEWS . . . 141

6.1 OV-1 High level operational concept graphic . . . 147
6.1.1 Graph model . . . 149
6.1.2 Probability model . . . 151
6.1.3 System dynamics model . . . 152
6.1.4 Markov chain model . . . 153
6.1.5 Petri net model . . . 154
6.1.6 Queueing model . . . 155
6.1.7 Discrete event model . . . 155
6.1.8 Agent-based model . . . 157
6.2 OV-2 Operational resource flow description . . . 158
6.2.1 Graph model . . . 159
6.2.2 Probability model . . . 161
6.2.3 System dynamics model . . . 162
6.2.4 Markov chain model . . . 162
6.2.5 Petri net model . . . 163
6.2.6 Queueing model . . . 164
6.2.7 Discrete event model . . . 164
6.2.8 Agent-based model . . . 165
6.3 OV-3 Operational resource flow matrix . . . 165
6.3.1 Graph model . . . 166
6.3.2 Probability model . . . 167
6.3.3 System dynamics model . . . 167
6.3.4 Markov chain model . . . 167

6.3.5 Petri net model 168
6.3.6 Queueing model 168
6.3.7 Discrete event model 169
6.3.8 Agent-based model 169
6.4 OV-4 Organizational relationships chart 170
6.4.1 Graph model 171
6.4.2 Probability model 172
6.4.3 System dynamics model 172
6.4.4 Markov chain model 173
6.4.5 Petri net model 176
6.4.6 Queueing model 176
6.4.7 Discrete event model 176
6.4.8 Agent-based model 177
6.5 OV-5a Operational activity decomposition tree 178
6.5.1 Graph model 179
6.5.2 Probability model 179
6.5.3 System dynamics model 180
6.5.4 Markov chain model 180
6.5.5 Petri net model 180
6.5.6 Queueing model 181
6.5.7 Discrete event model 181
6.5.8 Agent-based model 181
6.6 OV-5b Operational activity model 182
6.6.1 Graph model 184
6.6.2 Probability model 185
6.6.3 System dynamics model 186
6.6.4 Markov chain model 187
6.6.5 Petri net model 188
6.6.6 Queueing model 189

6.6.7 Discrete event model 190
6.6.8 Agent-based model 191
6.7 OV-6a Operational rules model 191
6.7.1 Graph model 194
6.7.2 Probability model 194
6.7.3 System dynamics model 194
6.7.4 Markov chain model 195
6.7.5 Petri net model 195
6.7.6 Queueing model 197
6.7.7 Discrete event model 197
6.7.8 Agent-based model 198
6.8 OV-6b State transition description 199
6.8.1 Graph model 200
6.8.2 Probability model 200
6.8.3 System dynamics model 201
6.8.4 Markov chain model 202
6.8.5 Petri net model 203
6.8.6 Queueing model 204
6.8.7 Discrete event model 204
6.8.8 Agent-based model 205
6.9 OV-6c Event-trace description 205
6.9.1 Graph model 207
6.9.2 Probability model 208
6.9.3 System dynamics model 208
6.9.4 Markov chain model 209
6.9.5 Petri net model 209
6.9.6 Queueing model 211
6.9.7 Discrete event model 211
6.9.8 Agent-based model 212

VII MODELING POTENTIAL OF SYSTEM ARCHITECTURE VIEWS 213

7.1 SV-1 Systems interface description 213

7.1.1 Graph model 215

7.1.2 Probability model 217

7.1.3 System dynamics model 217

7.1.4 Markov chain model 219

7.1.5 Petri net model 219

7.1.6 Queueing model 220

7.1.7 Discrete event model 220

7.1.8 Agent-based model 222

7.2 SV-2 Systems resource flow description 223

7.2.1 Graph model 225

7.2.2 Probability model 227

7.2.3 System dynamics model 227

7.2.4 Markov chain model 228

7.2.5 Petri net model 228

7.2.6 Queueing model 229

7.2.7 Discrete event model 230

7.2.8 Agent-based model 231

7.3 SV-3 Systems-to-systems matrix 232

7.3.1 Graph model 233

7.3.2 Probability model 233

7.3.3 System dynamics model 234

7.3.4 Markov chain model 234

7.3.5 Petri net model 235

7.3.6 Queueing model 236

7.3.7 Discrete event model 236

7.3.8 Agent-based model 237

7.4 SV-4 Systems functionality description 238

7.4.1 Graph model 240
7.4.2 Probability model 241
7.4.3 System dynamics model 241
7.4.4 Markov chain model 242
7.4.5 Petri net model 243
7.4.6 Queueing model 245
7.4.7 Discrete event model 246
7.4.8 Agent-based model 247
7.5 SV-5a Operational activity to systems function traceability matrix . 248
7.5.1 Graph model 249
7.5.2 Probability model 251
7.5.3 System dynamics model 251
7.5.4 Markov chain model 251
7.5.5 Petri net model 252
7.5.6 Queueing model 252
7.5.7 Discrete event model 252
7.5.8 Agent-based model 253
7.6 SV-5b Operational activity to systems traceability matrix 253
7.6.1 Graph model 256
7.6.2 Probability model 257
7.6.3 System dynamics model 257
7.6.4 Markov chain model 257
7.6.5 Petri net model 258
7.6.6 Queueing model 258
7.6.7 Discrete event model 258
7.6.8 Agent-based model 259
7.7 SV-6 Systems resource flow matrix 260
7.7.1 Graph model 261
7.7.2 Probability model 262

7.7.3 System dynamics model 263
7.7.4 Markov chain model 263
7.7.5 Petri net model 264
7.7.6 Queueing model 265
7.7.7 Discrete event model 266
7.7.8 Agent-based model 266
7.8 SV-7 Systems measures matrix 267
7.8.1 Graph model 270
7.8.2 Probability model 270
7.8.3 System dynamics model 271
7.8.4 Markov chain model 271
7.8.5 Petri net model 271
7.8.6 Queueing model 272
7.8.7 Discrete event model 272
7.8.8 Agent-based model 272
7.9 SV-8 Systems evolution description 273
7.9.1 Graph model 274
7.9.2 Probability model 274
7.9.3 System dynamics model 274
7.9.4 Markov chain model 275
7.9.5 Petri net model 275
7.9.6 Queueing model 276
7.9.7 Discrete event model 276
7.9.8 Agent-based model 276
7.10 SV-9 Systems technology & skills forecast 277
7.10.1 Graph model 278
7.10.2 Probability model 279
7.10.3 System dynamics model 279
7.10.4 Markov chain model 280

7.10.5 Petri net model 280
7.10.6 Queueing model 280
7.10.7 Discrete event model 281
7.10.8 Agent-based model 281
7.11 SV-10a Systems rules model 282
7.11.1 Graph model 282
7.11.2 Probability model 283
7.11.3 System dynamics model 283
7.11.4 Markov chain model 284
7.11.5 Petri net model 284
7.11.6 Queueing model 285
7.11.7 Discrete event model 285
7.11.8 Agent-based model 286
7.12 SV-10b Systems state transition description 287
7.12.1 Graph model 288
7.12.2 Probability model 289
7.12.3 System dynamics model 289
7.12.4 Markov chain model 290
7.12.5 Petri net model 291
7.12.6 Queueing model 291
7.12.7 Discrete event model 292
7.12.8 Agent-based model 293
7.13 SV-10c Systems event-trace description 294
7.13.1 Graph model 295
7.13.2 Probability model 296
7.13.3 System dynamics model 297
7.13.4 Markov chain model 297
7.13.5 Petri net model 298
7.13.6 Queueing model 300

7.13.7 Discrete event model 301
7.13.8 Agent-based model 301
VIII EXPERIMENTAL TESTING OF THE ELEMENT MAPS . . . 303
8.1 Experimental setup 303
8.2 2012–2013 Real World Design Challenge State Aviation Problem . . 307
8.2.1 Creating models from RWDC architecture views 311
8.3 2011 National Airspace System Enterprise Architecture Framework . 326
8.3.1 NASEAF As-Is Architecture 328
8.3.2 NASEAF Near Term Architecture 353
8.3.3 NASEAF Far Term Architecture 363
8.4 Discussion on the Research Arguments 368
IX CONCLUSIONS AND THE SOLSTYSS METHODOLOGY . . 369
9.1 Support for the first research argument 369
9.2 Support for the second research argument 371
9.3 Best modeling types for each viewpoint 375
9.4 Best viewpoint to develop for each modeling type 377
9.5 Recommended work flow for modeling systems of systems using their architectures 380
9.6 Future work 391
9.7 Final conclusions 395
APPENDIX A — ARCHITECTURE ELEMENTS TO MODEL ELEMENTS MAPS 396
APPENDIX B — ALGORITHMS USED IN THE EXPERIMENTS 401
REFERENCES 424
VITA 440

LIST OF TABLES

1 Operational view elements 143
2 System view elements 144
3 Model elements 146
4 Mapping between OV-1 and graph model elements 150
5 Mapping between OV-1 and probability model elements 152
6 Mapping between OV-1 and system dynamics model elements 152
7 Mapping between OV-1 and Markov chain model elements 154
8 Mapping between OV-1 and Petri net model elements 155
9 Mapping between OV-1 and queueing model elements 155
10 Mapping between OV-1 and discrete event model elements 157
11 Mapping between OV-1 and agent-based model elements 158
12 Mapping between OV-2 and graph model elements 160
13 Mapping between OV-2 and probability model elements 161
14 Mapping between OV-2 and system dynamics model elements 162
15 Mapping between OV-2 and Markov chain model elements 163
16 Mapping between OV-2 and Petri net model elements 164
17 Mapping between OV-2 and queueing model elements 164
18 Mapping between OV-2 and discrete event model elements 165
19 Mapping between OV-2 and agent-based model elements 165
20 Mapping between OV-3 and graph model elements 167
21 Mapping between OV-3 and probability model elements 167
22 Mapping between OV-3 and system dynamics model elements 167
23 Mapping between OV-3 and Markov chain model elements 168
24 Mapping between OV-3 and Petri net model elements 168
25 Mapping between OV-3 and queueing model elements 168
26 Mapping between OV-3 and discrete event model elements 169
27 Mapping between OV-3 and agent-based model elements 169

28 Mapping between OV-4 and graph model elements 172
29 Mapping between OV-4 and probability model elements 172
30 Mapping between OV-4 and system dynamics model elements 173
31 Mapping between OV-4 and Markov chain model elements 174
32 Mapping between OV-4 and Petri net model elements 176
33 Mapping between OV-4 and queueing model elements 176
34 Mapping between OV-4 and discrete event model elements 176
35 Mapping between OV-4 and agent-based model elements 177
36 Mapping between OV-5a and graph model elements 179
37 Mapping between OV-5a and probability model elements 180
38 Mapping between OV-5a and system dynamics model elements 180
39 Mapping between OV-5a and Markov chain model elements 180
40 Mapping between OV-5a and Petri net model elements 181
41 Mapping between OV-5a and queueing model elements 181
42 Mapping between OV-5a and discrete event model elements 181
43 Mapping between OV-5a and agent-based model elements 182
44 Mapping between OV-5b and graph model elements 184
45 Mapping between OV-5b and probability model elements 186
46 Mapping between OV-5b and system dynamics model elements 186
47 Mapping between OV-5b and Markov chain model elements 187
48 Mapping between OV-5b and Petri net model elements 189
49 Mapping between OV-5b and queueing model elements 189
50 Mapping between OV-5b and discrete event model elements 190
51 Mapping between OV-5b and agent-based model elements 191
52 Mapping between OV-6a and graph model elements 194
53 Mapping between OV-6a and probability model elements 194
54 Mapping between OV-6a and system dynamics model elements 195
55 Mapping between OV-6a and Markov chain model elements 195
56 Mapping between OV-6a and Petri net model elements 196

57 Mapping between OV-6a and queueing model elements 197
58 Mapping between OV-6a and discrete event model elements 198
59 Mapping between OV-6a and agent-based model elements 199
60 Mapping between OV-6b and graph model elements 200
61 Mapping between OV-6b and probability model elements 201
62 Mapping between OV-6b and system dynamics model elements 202
63 Mapping between OV-6b and Markov chain model elements 202
64 Mapping between OV-6b and Petri net model elements 203
65 Mapping between OV-6b and queueing model elements 204
66 Mapping between OV-6b and discrete event model elements 205
67 Mapping between OV-6b and agent-based model elements 205
68 Mapping between OV-6c and graph model elements 208
69 Mapping between OV-6c and probability model elements 208
70 Mapping between OV-6c and system dynamics model elements 209
71 Mapping between OV-6c and Markov chain model elements 209
72 Mapping between OV-6c and Petri net model elements 210
73 Mapping between OV-6c and queueing model elements 211
74 Mapping between OV-6c and discrete event model elements 212
75 Mapping between OV-6c and agent-based model elements 212
76 Mapping between SV-1 and graph model elements for both connection- and membership-oriented analysis. A single graph model can only be one of the either type, not both. 216
77 Mapping between SV-1 and probability model elements 217
78 Mapping between SV-1 and system dynamics model elements 218
79 Mapping between SV-1 and Markov chain model elements 219
80 Mapping between SV-1 and Petri net model elements 220
81 Mapping between SV-1 and queueing model elements 220
82 Mapping between SV-1 and discrete event model elements 222
83 Mapping between SV-1 and agent-based model elements 222

84 Modified Mapping between SV-1 and agent-based model elements. The changes were made in light of the attempt at modeling the FAA's Near Term Architecture. 223

85 Mapping between SV-2 and graph model elements for the more desirable option without phantom elements 227

86 Mapping between SV-2 and probability model elements 227

87 Mapping between SV-2 and system dynamics model elements 228

88 Mapping between SV-2 and Markov chain model elements 228

89 Mapping between SV-2 and Petri net model elements 229

90 Mapping between SV-2 and queueing model elements 230

91 Mapping between SV-2 and discrete event model elements 231

92 Mapping between SV-2 and agent-based model elements 231

93 Modified mapping between SV-2 and agent-based model elements . . 231

94 Example SV-3 constructed from Jones Wyatt's example problem [111]. 233

95 Mapping between SV-3 and graph model elements 233

96 Mapping between SV-3 and probability model elements 234

97 Mapping between SV-3 and system dynamics model elements 234

98 Mapping between SV-3 and Markov chain model elements 235

99 Mapping between SV-3 and Petri net model elements 235

100 Mapping between SV-3 and queueing model elements 236

101 Mapping between SV-3 and discrete event model elements 237

102 Mapping between SV-3 and agent-based model elements 237

103 Mapping between SV-4 and graph model elements 241

104 Mapping between SV-4 and probability model elements 241

105 Mapping between SV-4 and system dynamics model elements 242

106 Mapping between SV-4 and Markov chain model elements 243

107 Mapping between SV-4 and Petri net model elements 244

108 Mapping between SV-4 and queueing model elements 246

109 Mapping between SV-4 and discrete event model elements 247

110 Mapping between SV-4 and agent-based model elements 248

111 Example SV-5a for a generic reconnaissance UAV operation 249
112 Mapping between SV-5a and graph model elements 250
113 Mapping between SV-5a and probability model elements 251
114 Mapping between SV-5a and system dynamics model elements 251
115 Mapping between SV-5a and Markov chain model elements 251
116 Mapping between SV-5a and Petri net model elements 252
117 Mapping between SV-5a and queueing model elements 252
118 Mapping between SV-5a and discrete event model elements 253
119 Mapping between SV-5a and agent-based model elements 253
120 Example SV-5b constructed from author's previous work [20] 255
121 Mapping between SV-5b and graph model elements 257
122 Mapping between SV-5b and probability model elements 257
123 Mapping between SV-5b and system dynamics model elements 257
124 Mapping between SV-5b and Markov chain model elements 258
125 Mapping between SV-5b and Petri net model elements 258
126 Mapping between SV-5b and queueing model elements 258
127 Mapping between SV-5b and discrete event model elements 259
128 Mapping between SV-5b and agent-based model elements 259
129 Example SV-6 constructed from Jones Wyatt's example problem [111]. 261
130 Mapping between SV-6 and graph model elements 262
131 Mapping between SV-6 and probability model elements 262
132 Mapping between SV-6 and system dynamics model elements 263
133 Mapping between SV-6 and Markov chain model elements 264
134 Mapping between SV-6 and Petri net model elements 265
135 Mapping between SV-6 and queueing model elements 266
136 Mapping between SV-6 and discrete event model elements 266
137 Mapping between SV-6 and agent-based model elements 267
138 Modified mapping between SV-6 and agent-based model elements . . 267

139 Example SV-7 taken partially from the author's previous work on a system of systems performing a suppression of enemy air defenses mission [20]. 270
140 Mapping between SV-7 and graph model elements 270
141 Mapping between SV-7 and probability model elements 271
142 Mapping between SV-7 and system dynamics model elements 271
143 Mapping between SV-7 and Markov chain model elements 271
144 Mapping between SV-7 and Petri net model elements 272
145 Mapping between SV-7 and queueing model elements 272
146 Mapping between SV-7 and discrete event model elements 272
147 Mapping between SV-7 and agent-based model elements 273
148 Mapping between SV-8 and graph model elements 274
149 Mapping between SV-8 and probability model elements 274
150 Mapping between SV-8 and system dynamics model elements 274
151 Mapping between SV-8 and Markov chain model elements 275
152 Mapping between SV-8 and Petri net model elements 275
153 Mapping between SV-8 and queueing model elements 276
154 Mapping between SV-8 and discrete event model elements 276
155 Mapping between SV-8 and agent-based model elements 277
156 Mapping between SV-9 and graph model elements 278
157 Mapping between SV-9 and probability model elements 279
158 Mapping between SV-9 and system dynamics model elements 280
159 Mapping between SV-9 and Markov chain model elements 280
160 Mapping between SV-9 and Petri net model elements 280
161 Mapping between SV-9 and queueing model elements 281
162 Mapping between SV-9 and discrete event model elements 281
163 Mapping between SV-9 and agent-based model elements 281
164 Mapping between SV-10a and graph model elements 283
165 Mapping between SV-10a and probability model elements 283

166 Mapping between SV-10a and system dynamics model elements . . . 284
167 Mapping between SV-10a and Markov chain model elements 284
168 Mapping between SV-10a and Petri net model elements 285
169 Mapping between SV-10a and queueing model elements 285
170 Mapping between SV-10a and discrete event model elements 286
171 Mapping between SV-10a and agent-based model elements 287
172 Mapping between SV-10b and graph model elements 289
173 Mapping between SV-10b and probability model elements 289
174 Mapping between SV-10b and system dynamics model elements . . . 290
175 Mapping between SV-10b and Markov chain model elements 291
176 Mapping between SV-10b and Petri net model elements 291
177 Mapping between SV-10b and queueing model elements 292
178 Mapping between SV-10b and discrete event model elements 293
179 Mapping between SV-10b and agent-based model elements 293
180 Mapping between SV-10c and graph model elements 296
181 Mapping between SV-10c and probability model elements 297
182 Mapping between SV-10c and system dynamics model elements . . . 297
183 Mapping between SV-10c and Markov chain model elements 298
184 Mapping between SV-10c and Petri net model elements 300
185 Mapping between SV-10c and queueing model elements 300
186 Mapping between SV-10c and discrete event model elements 301
187 Mapping between SV-10c and agent-based model elements 302
188 Jones Wyatt's partial SV-7 based on her UAV design from the options provided by the RWDC problem (reproduced from her thesis[111]) . . 309
189 The viewpoints developed for Jones Wyatt's RWDC architecture. . . 310
190 Observations from the Jones Wyatt's RWDC architecture 310
191 The viewpoints developed for the three FAA NASEAF architectures. 327
192 Observations from the as-is version of NASEAF 329

193 An OV-2 viewpoint from the FAA as-is architecture translated to an adjacency matrix 347

194 An OV-2 viewpoint from the FAA as-is architecture translated to an adjacency matrix 348

195 Transition matrix made from the OV-5b of FAA's as-is architecture . 349

196 Observations from the near-term version of NASEAF 354

197 The origin of the information used to build an agent-based simulation 357

198 Observations from the far-term version of NASEAF 364

199 Probabilities used in the model from the FAA Far Term Architecture 365

200 Calculated communications probabilities in the FAA Far Term model. Disclaimer: the numbers are based on assumed probabilities. 366

201 Missing system of systems aspects in each modeling type 374

202 Best modeling types for each architecture viewpoint 376

203 Best viewpoint for each modeling type 378

204 Element maps for Graph, Probability, and System Dynamics models . 397

205 Element maps for Petri Net and Queueing models 398

206 Element maps for Markov Chain and Agent-based models 399

207 Element maps for Discrete Event models 400

LIST OF FIGURES

1 Overall flow of the document with key sections and chapters highlighted. 9

2 Decomposition of system of systems simulations 11

3 Georgia Tech Integrated Product and Process Development Methodology [174] (Reproduced) 12

4 Context is set. The next sections provide the background to the research. 15

5 Associative arrays store values linked to corresponding keys 25

6 Mapping between the set of designs and the set of representational models 26

7 Free body diagrams decompose the problem into more focused views . 30

8 Simulations map a system's description to its predicted performance . 33

9 Ease of engineering changes decreases rapidly during conceptual design [68] (Reproduced) 41

10 The effort required to make changes to software grows rapidly throughout the life-cycle [34] (Reproduced) 42

11 Research goals are set. The following is the formal development of the research arguments. 44

12 A Venn diagram that represents the idea of architectures capturing all aspects of systems of systems and a model capturing all aspects of the architectures 51

13 A Venn diagram that represents the idea of architectures capturing most aspects of systems of systems and a model capturing all aspects of the architectures 52

14 Method of picking a system of systems simulation type 58

15 Prandtl's comparison between induced drag and experimental data [184] 76

16 Working definition of a model and types of computer models to be investigated next. 79

17 Number of interactions between objects grow very fast 81

18 An example of a computer model evolution 91

19 An example of a computer model evolution 92

20 Georgia Tech Integrated Product and Process Development Methodology [174] (Reproduced) 99

21 Modeling Taxonomy adapted from Burbank et al.[42] 102
22 A mathematical graph 103
23 Scheduling with graphs 104
24 Pathfinding with graphs 104
25 Trees with graphs 105
26 Attributed connectivity with graphs (Map source: Wikimedia) 105
27 Conditional probability example: aircraft control surface actuation system 107
28 Combination of probabilities in the control surface actuation example 108
29 Series and parallel layout and possible combinations thereof 109
30 Recursive calculation scheme for series and parallel networks 110
31 Example probability network corresponding to the Python code . . . 111
32 An example continuous Markov chain 113
33 Example Markov chain transitions for a single state 115
34 Example Markov chain network 117
35 Example system dynamics network that represents an epidemic . . . 120
36 The equation for a level with an input and an output flow 121
37 Discrete-time Markov chain transitions 124
38 An example Petri net graph 127
39 Example Petri net graph in bi-partite arrangement 127
40 A Petri net model representing methane burning 128
41 Monte Carlo simulation for the integral of e^{-x} between 0 and 1 129
42 Monte Carlo simulation step for a discrete time Markov chain 131
43 Example queueing model 133
44 A block diagram of a simple discrete event simulator 137
45 A simplified schematic of an agent-based model 140
46 This chapter and next are used to go though each architecture view and determine their modeling potential. 145
47 Example OV-1 149

48 Example OV-1 to graph model translation 150
49 Example OV-1 to probability model translation 151
50 Example OV-1 to system dynamics model translation 153
51 Example OV-1 to Markov chain model translation 153
52 A part of an example OV-1 to Petri net model translation 154
53 A part of an example OV-1 to discrete event model translation. The OV-1 was taken from Joint Publication 4-09 Distribution Operations[186]. 156
54 A part of an example OV-1 to agent-based model translation 158
55 A part of an example OV-2 to graph model translation. The original OV-2 is on the top left. 160
56 A part of an example OV-2 to probability model translation 161
57 A part of an example OV-2 to Markov chain model translation 163
58 A part of an example OV-3 table to graph model translation. 166
59 Example OV-4. Shaded region will be used in the examples. 171
60 Example OV-4 to graph model translation 172
61 Example OV-4 to Markov chain model translation 175
62 Example OV-4 to agent-based model translation 178
63 An example OV-5b developed from Marine Corps Warfighting Publication for Suppression of Enemy Air Defenses[60]. 183
64 An example OV-5b to graph model transformation based on the OV-5b shown in Figure 63. 185
65 An example OV-5b to probability model transformation based on the OV-5b shown in Figure 63. 185
66 An example OV-5b to system dynamics model transformation based on the OV-5b shown in Figure 63. 187
67 An example OV-5b to Petri net model transformation based on the OV-5b shown in Figure 63. 188
68 An example OV-5b to queueing model transformation based on the OV-5b shown in Figure 63. 189
69 An example OV-5b to discrete event model transformation based on the OV-5b shown in Figure 63. 190

70 A representative text that can be found in a OV-6a. The three sentences above were taken as examples from Standing Rules of Engagement for U.S. Forces[47]. 193

71 A representative pseudocode that can be used as an OV-6a (Reproduced from the work of Mittal et al.[141]). Equation 65 shows the math notation for this pseudocode. 193

72 An *if..., then...* statement as a Petri net model. 196

73 An example OV-6a to Petri net model transformation based on the OV-6a Rule 1 given in Figure 70. Colors 1–3 represent hostile intent, hostile act, and neutral entities. States P, H, and F represent peaceful, hostility, and force use states. 196

74 An example OV-6a to agent-based model transformation based on the OV-6a Rule 3 given in Figure 70. The solid lines and dashed lines represent different scenarios. 198

75 An example OV-6b developed from Marine Corps Warfighting Publication for Suppression of Enemy Air Defenses[60] for how the enemy is expected to operate. 200

76 An example system dynamics model translated from the OV-6b shown in Figure 75. 201

77 An example Markov chain model translated from the OV-6b shown in Figure 75. 202

78 An example Petri net model translated from the OV-6b shown in Figure 75. 203

79 An example discrete event model translated from the OV-6b shown in Figure 75. 204

80 An example OV-6c depicting the operation of a coffee machine in an office. 207

81 An example OV-6c to Petri net model transformation based on the OV-6c shown in Figure 80. 210

82 An example OV-6c to discrete event model transformation based on the OV-6c shown in Figure 80. 212

83 SV-1 reproduced from Jones Wyatt et al.'s example [110] with system node additions 214

84 Example SV-1-to-graph model translation options based on Figure 83 216

85 Example high fidelity system dynamics model based on the SV-1 given in Figure 83. 218

86 Example low fidelity system dynamics model based on the SV-1 given in Figure 83. 218

87 Example SV-1-to-agent-based model translation based on the SV-1 given in Figure 83. 223

88 SV-2 reproduced from Jones Wyatt et al.'s example [110] with system node additions 225

89 Example SV-2 to graph model translation based on the SV-2 given in Figure 88 that introduces phantom edges. The edges shown with dashed lines do not exist in the architecture but are necessary in the graph model. Print and optical disk exchanges are depicted as *physical* and wired and wireless network data transfers are depicted as *network*. 226

90 Example SV-2-to-graph model translations based on the SV-2 given in Figure 88 that is free of phantom edges. The graph on the left can be rearranged into the one on the right to highlight the bipartite nature of the resulting graph model. 226

91 Example SV-2 to system dynamics model translation based on the SV-2 given in Figure 88. Here, only the exchanges between the Windows server and Linux server is shown for clarity. 228

92 Example SV-3-to-discrete event model translation based on the SV-3 given in Table 94. 237

93 Example SV-3-to-agent-based model translation based on the SV-3 given in Table 94. 238

94 Example SV-4 depicting the interoperation of trains and buses for a public transportation system. Passenger parameters box is a data store and contains all passengers waiting in all stops and stations. 239

95 Example SV-4 to graph translation based on Figure 94. 241

96 Example SV-4 to Petri net model translation based on Figure 94. . . 244

97 Swapping the places and transitions is possible; however, makes the mapping much less crisp and results in a larger model that is more difficult to simulate 245

98 Example SV-4-to-agent-based model translation based on Figure 94. . 247

99 How the SV-4, -5a, and -5b are interlinked 249

100 Example SV-5a to graph model based on the SV-5a given in Table 111. 250

101 Example SV-5b to graph model based on the SV-5b given in Table 120. 256

102 Three example of SV-6-to-graph model translation. The source is Table 129 constructed from Jones Wyatt's thesis [111]. 262

103 Example SV-6-to-system dynamics model translation based on the SV-6 given in Table 129. 263

104 Example SV-6-to-Petri net model translation based on the SV-6 given in Table 129. 265

105 An example SV-9 depicting technology development for commercial aircraft based on NASA's Environmentally Responsible Aircraft program[150].278

106 Example SV-9-to-graph model translation based on the SV-9 given in Figure 105. Additional information about what each of the subprojects deal with are filled in using expert knowledge. This information could be easily included in an SV-9. 279

107 A representative text that can be found in an SV-10a. A similar rule was used in the author's earlier work on a system of systems designed for suppression of enemy air defenses mission [20]. 282

108 Example modifications to a discrete event model based on an SV-10a rule 286

109 An example SV-10b depicting the operation of a train. It shows more detail about the train system compared with the SV-4 given in Figure 94.288

110 Example SV-10b-to-discrete event model translation based on the SV-10b given in Figure 109. 292

111 Example SV-10b-to-agent-based model translation based on the SV-10b given in Figure 109. Notice that the environment is represented with dashed lines because it is missing in the SV-10b. 294

112 An example SV-10c depicting the operation of a train. It shows significantly more detail compared with the SV-4 given in Figure 94. . . . 295

113 Example SV-10c-to-Markov chain model translation based on the SV-10c given in Figure 112. This model only depicts the control system in isolation from other subsystems, which is not realistic and can only be used with limited success. If other subsystems are to be included in the model, the overlapping states must be split into numerous states as only one Markov state can be active at any given time as discussed in the text. 298

114 Example SV-10c-to-agent-based model translation based on the SV-10c given in Figure 112. This model only depicts the internal logic for the operator agent within the train system. The agent communicates with the environment and control system only because the SV-10c does not show any subsystems interacting with the operator. 302

115 This chapter tests the element maps created in the last two chapters. 306

116 Jones Wyatt's SV-1[111] for the RWDC problem (Adapted from [51]) 308

117 One of the several infinite loops that arise in the agent-based model. . 315

118 The simulation ends immediately after 100 time units. The data is based on 250 repetitions. 319

119 Roughly 25–30 future events are kept in the list thanks to the queuing behavior. The data is based on the maximum number of events in the list for 250 repetitions. 320

120 The queues grow linearly over time with no limit. The graph shows minimum, mean, and maximum queue lengths at each simulation step over 250 repetitions. 321

121 A graphical representation of the Real World Design Competition System Dynamics model 323

122 Selected results of the Real World Design Competition System Dynamics model. There are more stocks in the model but they are left out for clarity. 324

123 A part of the NASEAF as-is OV-5b 330

124 A possible translation of the OV-5b to a discrete event model 331

125 A possible translation of the OV-5b to a Markov chain model 332

126 The resulting Markov chain output for the probabilities of each state being active. 333

127 The resulting discrete event output for the probabilities of each server being busy. 333

128 The resulting Markov chain output for the probabilities of each state being active. 334

129 The resulting discrete event output for the probabilities of each server being busy. 335

130 FAA's As-Is OV-6c (fourth out of five views) 339

131 FAA's As-Is OV-5b with added sequence information from the OV-6c (adapted from the FAA architecture [19]) 340

132 FAA's As-Is OV-5b with added sequence information from the OV-6c (adapted from the FAA architecture [19]) 340

133 Agent-based model results using the FAA As-Is OV-6c 345

134 Agent-based model results using the FAA As-Is OV-6c and OV-5b. . 346

135 FAA's As-Is OV-2 (third out of three views) 351

136 The resulting Markov chain output for the probabilities of each of the five states being active. 352

137 FAA Mid Term Petri net model for weather (partial) 362

138 How parts of the SV-2 and SV-6 are used to create a probability model 365

139 Translating the FAA Far Term SV-6 to a discrete event model 367

140 Translating the FAA Far Term SV-6 to a discrete event model 368

141 How architectures become models 370

142 Architecture views that are instrumental in creation of multiple modeling 379

143 SOLSTySS flowchart has 5 different parts. Part A is the starting point and is always executed. Other parts are executed if they fit the scenario. 383

144 Starting point for the methodology. Scenarios are selected here. . . . 388

145 If no previous models exist, this part of the methodology is used. . . 389

146 If some alternative models are sought with no particular requirements, this part of the methodology is used. 390

147 If a cross-validation model is desired for one of the existing models, this part of the methodology is used. 391

148 In some cases, especially if views are not enough to do a specific kind of modeling, this supplementary part of the methodology is used. . . 392

LIST OF ACRONYMS AND ABBREVIATIONS

ABM Agent-based Model/Simulation.

AoA Analysis of Alternatives.

ATCSCC Air Traffic Control System Command Center.

DARPA Defense Advanced Research Projects Agency.

DES Discrete Event Simulation/Model.

DOD United States Department of Defense.

DoDAF Department of Defense Architecture Framework.

DOTMLPF Doctrine, Organization, Training, Materiel, Leadership, Personnel, Facilities.

FAA Federal Aviation Administration.

ICAO International Civil Aviation Organization.

IEC International Electrotechnical Commission.

IEEE Institute of Electrical and Electronics Engineers.

INCOSE International Council on Systems Engineering.

IPPD Integrated Product and Process Development.

ISO International Organization for Standardization.

JCIDS Joint Capabilities Integration and Development System.

MC Markov Chain.

NAS National Airspace System.

NASEAF National Airspace System Enterprise Architecture Framework.

PN Petri Net.

Prob Probability Model.

QT Queueing Theory/Model.

RWDC Real World Design Challenge.

SD System Dynamics.

SOLSTySS Selection Of Logical Simulation Types for Systems of Systems.

SoS System of systems.

SysML Systems Modeling Language.

TMU Traffic Management Unit.

UAV Uninhabited/Unmanned Aerial Vehicle.

UML Unified Modeling Language.

XML Extensible Markup Language.

GLOSSARY

bipartite graph A graph that can be split into two groups of vertices in a way that the vertices within each group are not adjacent to other vertices in the same group., p. 215.

conceptual model A modeler's understanding of how the world works translated into a formal language (graphical, text-based, or a combination of both)., p. vii.

system of systems architecture a collection of views that depict an order and a modus operandi for the constituent systems as well as the interactions between them, p. 45.

LIST OF SYMBOLS

:	Such that
$\#_Y(R, C)$	Count of number of "Yes" entries in the intersection of rows R and columns of C of the element map
$\mathcal{AR}$	Aspect ratio of an aerodynamic surface
$\cap$	Set intersection
$\cup$	Set union
$\underline{\underline{T}}$	Transition matrix in a Markov chain model
$\emptyset$	Empty set
$\exists$	There exists
$\forall$	For all
$\Longleftrightarrow$	If and only if
$\perp\!\!\!\perp$	Statistical independence
λ	Rate (of probability flow) in a Markov chain (based on an exponential distribution)
$\wedge$	And
$\mid$	Such that
$\Rightarrow$	Implies
$^L/_D$	Lift-to-drag ratio
$\supseteq$	Super set of, equal to

$\underline{V}(t)$ Time dependent state vector in a Markov chain model

B_M All modeling types that are determined to be the best modeling types for to the most detailed existing view

C Set of columns of the element map table

C^M Columns of the element map specific to the set of models M

c_D Coefficient of drag

c_L Coefficient of lift

c_{Di} Coefficient of induced drag

E Element map table

E Set of edges in a graph model

f_i Flow i in a system dynamics model

G Graph

M Computer model

$M^\star$ Set of already existing computer models

$P(A|B)$ Probability of event A occurring given that event B has occurred

$P(A)$ Probability of event A

R Set of rows of the element map table

R^V Rows of the element map specific to the set of architecture views V

S_x Stock named x in a system dynamics model

V Architecture view

V Set of vertices in a graph model

$V^{\star}$ Set of already existing architecture views

A^{c} Complement of a set A

SUMMARY

Engineering design is a cycle of decisions and analyses. Analysis of design alternatives provide the performance predictions for decisions to be made; decisions provide the context for analyses to be built. As decisions are made, the design gains detail that must be documented for communication purposes between engineering teams. These documents serve as the rules, logic, assumptions, metrics to build engineering analysis models. Especially in multi-disciplinary design activities, such documents form the consistent basis for the disciplinary modeling teams build representations for the system to perform analyses that support further decisions.

System of systems design takes significant effort in planning, engineering, and testing because they are formed via the integration of heterogeneous systems built and operated by multiple entities. Their design description documents are compiled and distributed among the entities, so that they can engineer effective systems that can cooperate smoothly inside the integrated system of systems. These design descriptions are critical in the process of designing, acquiring, training, operating, maintaining, and evolving activities necessary for systems of systems.

It is customary to use architectures for documentation purposes in the field of systems of systems. For analysis, subject matter experts or computer simulations are used. These methods are—or at the very least, becoming—industry standards and cover the entire life-cycle of the system of systems. However, there are a large number of possible analysis methods for systems of systems, and the process of building such analysis models from architectures is left to the modeler. Standards do exists for architectures and their computer representations but widely accepted ways of building

models from architectures do not exist.

Because there is no single, accepted, and widely-used system of systems modeling technique, for every problem, a decision also needs to be made on what type of model to use for system of systems analysis. Focusing on early design and planning phases of systems acquisition, this research deals with the links between the descriptions, models, and simulations of systems of systems. The main research question answered is how a design engineer can select the correct modeling technique—not the model, but its type—from a number of alternatives.

There are two main research arguments in this thesis. The first one is that *system architectures are very closely related to conceptual models* (abstracted understanding of systems). This argument is presented in an inductive form and subsequent experimental work details how this argument holds up to scrutiny. The second research argument is that *depending on the architecture views deemed to be essential to describe a system of systems, there are a number of modeling techniques required to adequately model it*, which is in a hypothesis form. This hypothesis was tested by assessing overlaps of coverage that architecture views and models exhibit. The two research arguments are tied together by the fact that conceptual models are an essential step in creating executable models that are simulated on a computer.

Element maps were created to test the hypotheses in a systematic way. These element maps link standard data types found on the viewpoints of the Department of Defense Architecture Framework (DoDAF) to standard modeling elements that make up computer models. DoDAF was selected due to its applicability to system of systems problems. The element maps provide a repeatable scaffold in the translation of architectures into executable models.

Using the element maps, eleven tests were performed on 4 different architectures. They vary in size and purpose. For each of the architectures a set of viewpoints were

investigated and executable models were produced. The process of going from architecture views to the code of the computer models is detailed to help system of systems engineers to make similar decisions. Using the element maps, future projects can determine what modeling type is more appropriate or what architecture definitions are missing to develop desired computer models. Therefore, the element maps are one of the main contributions of the work culminated from the testing of the research arguments and is used in the methodology for selecting the right modeling type from the existing architecture views. They are added to the toolbox of the system of systems engineer.

The consequences of the research arguments are also used to construct a useful methodology for system of systems modeling. The methodology is named Selection Of Logical Simulation Types for Systems of Systems (SOLSTySS). Equipped with the knowledge of which model(s) can offer adequate modeling for a given problem (function of SOLSTySS) and a means to translate architecture descriptions into conceptual or computer models (function of the Element Maps), the design engineers expedite the analysis process as well as introduce traceability into the process of modeling for systems of systems. The processes are detailed in the concluding chapter of the document. The SOLSTySS methodology helps the system of systems engineers select their models and provide them with a scaffold for further computer model development.

CHAPTER I

INTRODUCTION

> There can be no doubt that a tribe including many members who were always ready to give aid to each other, would be victorious over most other tribes; and this would be natural selection.
>
> Charles R. Darwin [55]

Collaboration is an essential natural process. Animals use collaboration to hunt, survive, build nests/colonies, and provide safety for their offspring. Human civilizations used collaboration to drive specialization in their economies, enable large scale planning execution of national policies, and use their natural resources more efficiently[1]. Technological limits constrained collaboration to remain a phenomenon between two organic systems; no human-made tool, system, or machine enjoyed the benefits of collaboration until the 20th century.

As machines are put together in a collaborative fashion, previously unthinkable capabilities are beginning to emerge from them (e.g., the Internet of things). Such changes to the way that systems are used can be thought of as disruptive technologies. Disruptive technologies create a market shift within the related industries compared with incremental technologies that improve a process within industries. In order to create the shift, these disruptive technologies rely on superior performance over their

[1] In some circles this collaboration is known as interoperability. For example NATO defines interoperability as *the ability to act together coherently, effectively, and efficiently to achieve Allied tactical, operational, and strategic objectives* [9]. Interoperability and cooperation have the same meaning and there is little difference to warrant the use of both nouns separately. Because collaboration is etymologically older, it will be used throughout.

traditional ones to convince the industry to abandon traditional, proven, and well-understood practices. However, making the jump to the new and not well-understood processes comes with risks for the industry.

The jump to the new methods and processes requires training in those new methods. It could be argued that designing collaborative systems that will render older systems obsolete requires a different set of skills and steps compared to classical systems design. The traditional practices must be augmented with new tools and theories well beyond its original scope to deal with this new type of industry practice, in this instance: *designing collaborative systems*.

Making systems collaborate efficiently is not an easy task to accomplish. The United States Department of Defense (DOD) as well as other government agencies throughout the world have been prioritizing cooperation issues in their system acquisitions to fulfill their information technology needs or enhance legacy systems giving them new capabilities [85]. Integration of heterogeneous systems into an artificial society, commonly referred to as a *system of systems*, takes significant effort in planning, engineering, and testing. The enhanced connectivity creates various collaborative dynamics between systems, which are often difficult to predict prior to their operation. In addition to the integration challenges, the systems themselves became more complicated and their designs take longer and cost more.

Unsurprisingly, new industries and businesses that were created to capitalize on this new paradigm are becoming very successful in the way they deal with complexity and emergence. Information technology giants such as Google and Facebook find very convincing uses for the increased connections, sharing, and collaboration between systems[2]. There are important lessons to be learned from such companies; however, these are commercial entities and have the freedom to move in or out of a market as

[2]Google revolutionized the web search by providing the service on a cluster of cheaper computers compared to a single powerful but expensive server machine [139].

they desire. Military organizations cannot make the decision of not protecting certain areas of the country. Having said that, they do have some doctrinal flexibility of how to protect the country: e.g., not every military force has to have tanks. If having tanks is not effective, that army can decide not to acquire or use tanks.

Historically, when a new capability/functionality was needed, this function was added on the system that worked in that area. For example, bomber aircraft were equipped with radars, cameras, and other sensors to find the targets and deliver their payload to a precise area rather than dropping a large amount of free-fall ordnance around the target. In this process of slowly adding more and more functionality, systems became larger, more complicated, more expensive, and harder to be modernized. In order to alleviate the growth of complication, different systems can be connected together performing different tasks. The rapidly developing communication technologies are enabling dispersed functionality across systems. This allows for less complicated individual systems but a more complicated whole, which may or may not be desirable. Today, the functions of systems are viewed as services to the group (e.g., Global Positioning System satellites provide a geo-location service). This approach centralizes the source of the service, but makes the benefits available to all compatible systems, hence allowing them to be more useful without adding many components to the system[3].

The spread of functions to different systems may make systems cheaper as a whole as well. Because there is a dedicated server for the function, the server is expected to be busy most of the time. Comparing that scenario with a function that exists on all systems, which are part of the system of systems, highlights the resources a distributed system wastes. Not every cell phone requires its own satellites

[3]Interestingly, Global Positioning System signal processing requires some difficult calculations to be performed on the receiver side. To combat the battery drain, it has been proposed that the devices should upload the signal they receive to a cloud computing platform, and download the results once they are processed, saving battery charge in the process [125]. This is a perfect example of locating functions away from the final user.

for navigation purposes, but they all require at least one antenna. This drive for efficiency and cost effectiveness is the main motivation behind system of systems engineering; in theory it makes better, more efficient, more effective, and more useful systems possible. However, decentralizing functions does not make the whole system less complex as the complexity is pushed to the communication network that must support the distributed functions.

System designers are also facing increasing complexity in their products. Such products take significantly longer to design, test, produce, and field [154, 33]. The DOD desires critical systems to be delivered for the warfighter's use much faster than today's acquisition system allows for. This is evidenced by the Secretary of Defense's choice of three focus areas: cost, schedule, and performance [11]. In parallel, Defense Advanced Research Projects Agency (DARPA) has been working on the META program with a goal for improving the design, integration/manufacturing, and verification of complex cyber-physical systems (i.e., physical systems that rely on software as part of their operation) with a special focus on aerospace systems [54]. It is clear that the reduction in design cycles is a priority for defense systems.

Correcting for the long design cycles, the military leaders look out for many long-term projects. The next fighters, warships, or tanks are not only designed to combat today's or near-term threats, they are designed to effectively combat futuristic possible threats. Some of these threats fail to materialize, which causes unnecessary resource drain. Also, the requirements are set to be very aggressive and, without the infusion of many yet-exotic technologies, a technically feasible design cannot be obtained. Such technologies take time to mature and their impacts are uncertain and in flux. All of these reasons cause costs to balloon; however, the programs are pursued until they cannot be sustained anymore and the increasing costs ultimately lead to cancellation. For example, the Future Combat System—a recent major modernization effort by the United States Army—had a 28% estimated chance to meet

its budget constraint and it was approved to continue development nevertheless [48]. Unsurprisingly, it was canceled later due to budget cutbacks.

1.1 Motivating problem

Given the issues and recommendations above, early phase design must be improved for defense systems. However, many modern defense acquisitions deal with systems of systems and the improvements in systems engineering should be reflected in system of systems engineering as well. One major difference between systems design and system of systems design is that the latter is not done from scratch. There usually is some existing system(s) that the new system will interface with. A system of systems design is, therefore, more similar to a spiral system development: a substantial but incremental upgrade to an already operational system of systems. However, the disruptive feature of systems of systems challenge simple spiral development methods. More suitable design methodologies exist that guide systems of systems design using iterative cycles [52, 90]. The evolutionary nature must play a central role in any analysis or design effort for systems of systems.

The analysis of alternatives (AoA) is the general method used by the DOD to check the *goodness* of a multitude of possible solutions. It is used to identify the most promising potential materiel solution [58]. The method is mainly used to assure that enough alternatives are considered before risky and costly materiel acquisition decisions are made. However, the process of AoA is manual and very slow. Defense Acquisition Guidebook states that evaluating too many alternatives is worse than evaluating too few, because too many alternatives exceeds the resources of the team performing the study [58]. Analyzing alternatives and making design decisions based on the results is a rigorous trial and error process, e.g., optimization or design space exploration. At each phase of the evolutionary design process, the engineers must evaluate different options and make a selection as well. This selection process must

be done in a fair, rigorous, and transparent way, which leads to a quantitative study. When design selection is not done properly [16], time [49] and resources are lost during the re-evaluation of alternatives. Because design AoA is analogous to DOD AoA, the analysis for design options must also be done in a fair, rigorous, and transparent way.

A necessary difference between the DOD AoA and design alternative evaluations is that the design counterpart must be done much more rapidly. More design alternatives are to be investigated compared with selecting an alternative among finished design proposals. In fact, the sheer number of architecture alternatives are prohibitive even for relatively simple architecture design spaces due to the existence of discrete variables [104]. Every discrete variable multiplies the number of possible designs by a factor of how many settings it has. Additionally, engineering design is an iterative process. The design evaluations not only serve as a way to select a final design, but also guide the next round of the design iteration. Therefore, the methods used within the AoA step during design, its planning, construction, and use, must be geared towards evolutionary designs.

The large number of alternatives to be evaluated forces the analysis to be automated. These automated analyses are usually computer simulations. Simulations use virtual representations of the system in question and predict how it would behave in the real world, much like forming hypotheses and performing experiments in the scientific sense. Since simulations are to be used as a part of the design process, the questions that deal with systems of systems design apply to simulations of them as well. Another opportunity for automation lies in the translation of design descriptions into executable computer models for analysis. Further development on the modeling and simulation and consistency/logic checks that integrate seamlessly with design descriptions is desired [59].

Based on the discussion above, a list of computer modeling questions for system of systems engineering is posed below. This list will serve as a guidance for the research

effort.

1. At which step of the design process should systems of system be simulated?
2. What should be the general steps of simulations for system of systems design?
3. What are the different options for each step of system of systems simulations?
4. Which options are more suitable for various kinds of systems of systems?
5. Once picked, how should the steps of simulations be executed?

The work performed here has the form of a methodology. The first three questions are mostly answered by a literature review of the field as well as similar fields. As such, the chapters discussing these topics mainly identify and synthesize existing information to answer the questions. New information is only added when necessary in the development of the methodology, e.g., author's interpretations of the observations and setting expectations for the methodology. The reader can expect mainly deductive arguments in these chapters. Conversely, the answer to the fourth question will include synthetic arguments (hypotheses and inductions), which will be verified with appropriate methods. The fourth question is then the most central to the research among the five questions; therefore, it is used as the main research question. The fifth question is used to bring the research to full circle in the sense of infusing the new knowledge into the design process for systems of systems.

Before answering the questions, however, the scope of the research must be set. The work performed here does not deal with all aspects of systems of systems engineering such as physical testing, acquisition, and operation after acquisition, which would make the topic very large in scope. The aspects to be dealt with are design, development, and iterative refinement through virtual testing. These topics mainly focus on integration issues and performance metrics such as assets lost/damaged, time to complete mission, network vulnerability, and operation cost. Nevertheless,

a good understanding of the system of systems concept is required for a simulation study; therefore, such aspects of systems of systems will be included in the study but not investigated in depth. The author argues that the models—similar to the ones that are investigated here—can be used to calculate metrics related to acquisition, testing, and operation. The development of such models, however, will be left as a related future expansion to the work detailed here. The overall goal of the work is given below.

Research Objective. Develop a methodical approach for selecting a modeling type for analysis of systems of systems, that is useful for early design phases as well as for continuous cycles of system of systems evolution.

The answers to the first two questions can be found in Chapter 2, which will also serve as a broad description of system of systems engineering. Chapter 3 includes thesis arguments and the overview of the experimental work. The third question is answered in Chapter 5, which is preceded with a chapter on models in general. The Chapters 6 and 7 detail the work that went into experimental setup and are followed by Chapter 8 where the tests and experiments are performed. Finally, Chapter 9 concludes the work by highlighting research objectives achieved. Figure 1 displays the overall flow of the document.

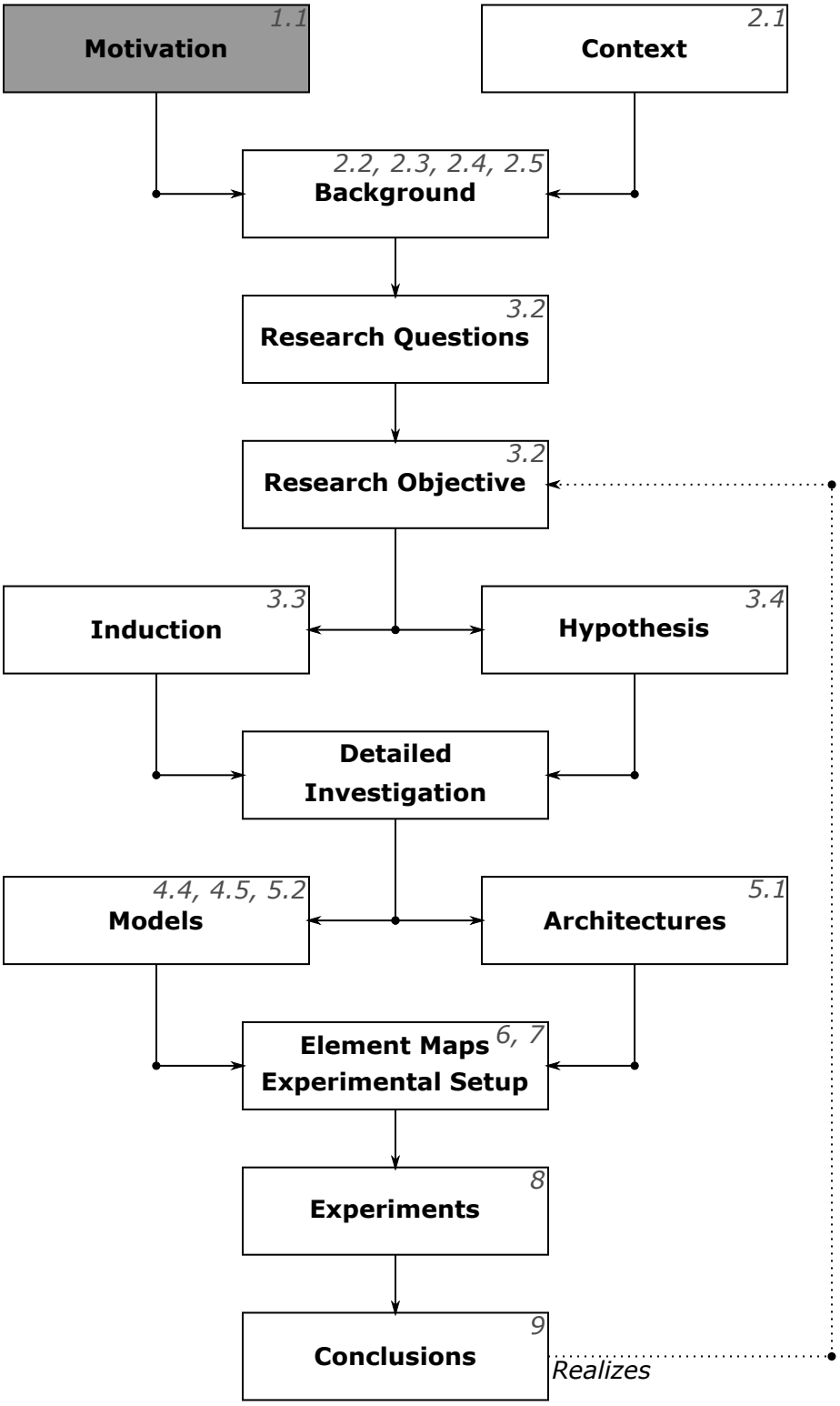


Figure 1: Overall flow of the document with key sections and chapters highlighted.

CHAPTER II

ON THE DESIGN OF SYSTEMS

> Incremental improvement is guaranteed to be obsolete over time. Especially in technology, where you know there's going to be non-incremental change.
>
> Larry Page [123]

The first research question deals with the role of simulations during system of systems design. Before this question can be answered, a discussion on systems of systems and their design is warranted. In this chapter, using the purest systems engineering approach, the study of system of systems will be broken down into its components. Some components of system of systems design are borrowed from different fields of mathematics, science, and engineering. This aggregation of many smaller problems and techniques must be understood and defined in useful ways to propose improved solutions to design problems. After the discussion of each component, they will be integrated into system of systems design. Finally, the warrant to use simulations in system of systems design will be justified. The difficulties and emerging problems discussed in this chapter will be used in Chapter 3 to formulate hypotheses and their tests. From the topic of system of systems simulation, the parts seen in Figure 2 emerge.

This chapter is intended to provide the reader with some basic understanding of each and every one of the parts shown above in Figure 2, to define them for further methodology development purposes, and to define central ideas in system of systems engineering.

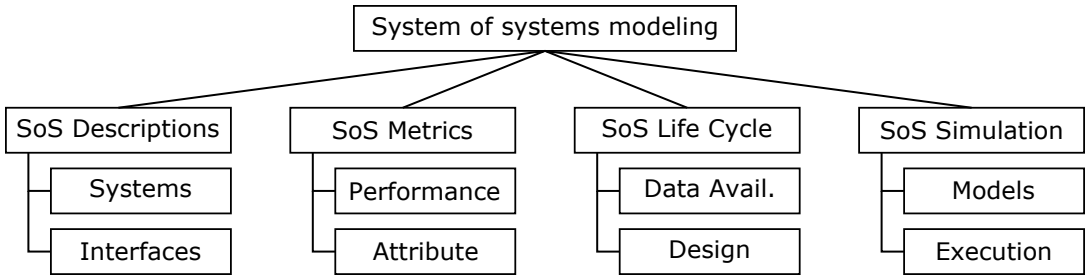


Figure 2: Decomposition of system of systems simulations

2.1 Engineering design

The design activity is composed of a chain of decisions [94, 179]. Along the way, the engineers make decisions that give the design its form and function. In order to make decisions, engineers rely on knowledge about the machines that they are building. In fact, the entire scientific body of knowledge is made to make such decisions [96]. Using knowledge about nature, we can make decisions that lead to better actions. Design decisions made throughout the design process result in a final design; therefore, the final design can be thought of as the aggregation of all design decisions.

The mechanics of decision making is best left to psychologists; however, many systematic, step-by-step processes for making decisions have been constructed and published in the context of engineering design [21, 174, 171]. These methods follow a top-down direction, which is a common characteristic of requirements engineering. Because they are very similar, and for the purposes of this work the differences are not critical, the author has elected to use the decision process within the Integrated Product and Process Development (IPPD) Methodology.

IPPD was developed at Georgia Institute of Technology [174] and is depicted in Figure 3. It includes six logical steps for design decision support as well as product design steps. IPPD is a generic enough method that one can apply to many engineering problems. The steps are sequenced to ensure that an organized and complete list of factors are considered before making a design decision. These steps are given below with descriptions in the context of system of systems engineering.

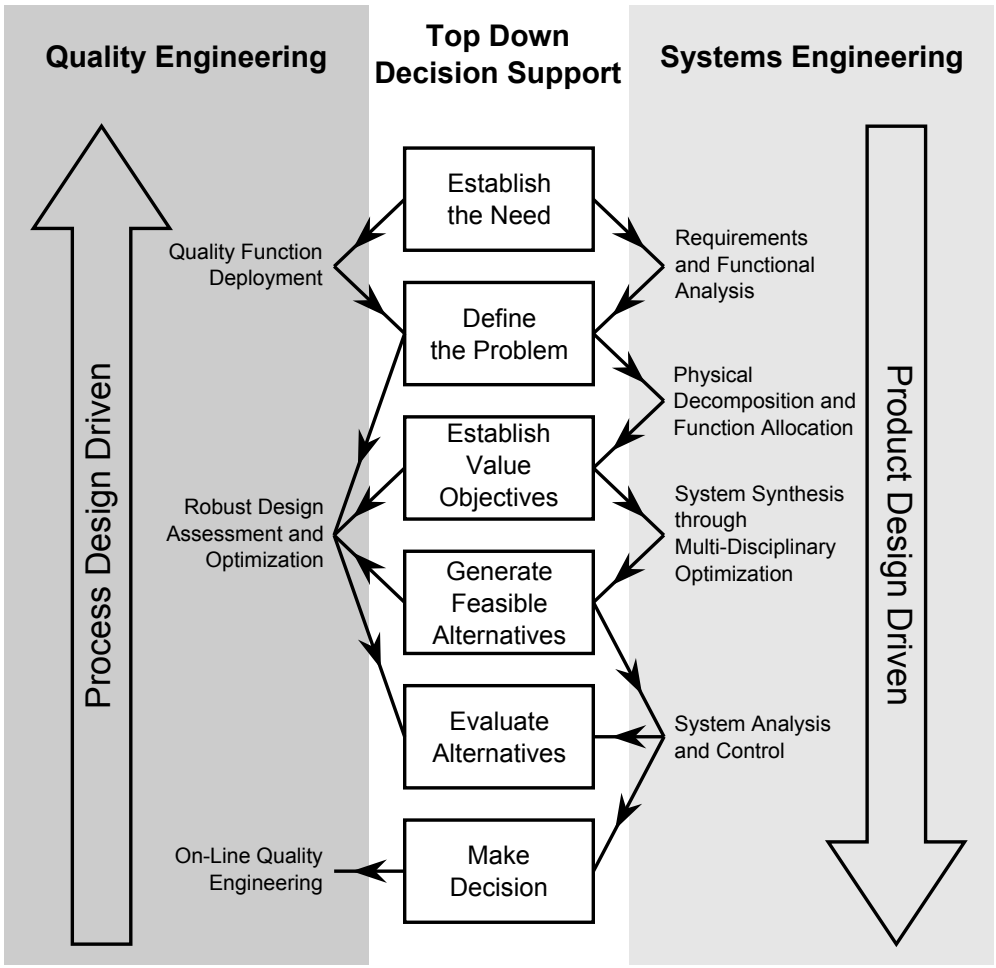


Figure 3: Georgia Tech Integrated Product and Process Development Methodology [174] (Reproduced)

1) Establish the need Discovering and enumerating capabilities identified as lacking currently.

2) Define the problem Bounding of the study to a limited number of goals derived from identified needs.

3) Establish value objectives Finding quantifiable measures of effectiveness that are to be reached. Reaching these objectives should guarantee achieving the goals set in the previous step.

4) Generate feasible alternatives Listing or defining a space of possible solutions that may exceed value objectives.

5) Evaluate alternatives Trying, evaluating, and selecting or discarding each alternative based on their quantified performance by the measures of effectiveness.

6) Make decision Selecting the best alternative from a final list of all successful alternatives, ordered by a preference function.

The first two steps are about defining a reason for the design activity itself. These steps can be supported by analysis; however, the analysis in question is on the existing systems and processes to highlight their shortcomings. This type of analysis is commonly referred to as a *gap analysis*. The second step bounds the problem by declaring the final goals of the design effort. While those goals can be compared to the already existing systems and be validated that they would fill the gaps identified earlier, the analysis here has few similarities with predicting the performance of a possible design.

The third step derives the important metrics of the problem. Which properties of the design are important? Is it the range of a bomber or the payload capacity that the customer values most? There is a very delicate link between this step and analysis. Ideally, the metric determination should be independent of the analysis methods, i.e., the engineering problem should be guided by the needs established beforehand. However, if the needs established earlier are not calculable, either needs must be modified or proxies to them must be established (e.g., mean time between failure instead of reliability, radar cross section instead of detectability). Therefore, existing analysis methods play a large role in determining the metrics for the design problem. The practicalities of analysis methods must keep decision makers grounded.

The fourth step, generating feasible alternatives, is intrinsically linked to analysis. Figuring out whether a design alternative is feasible or not requires analysis. In some design problems, feasibility is handled at the next step (i.e., evaluation) and only compatibility is taken into consideration during the fourth step. Generation of

alternatives is another step in which the possibilities and practicalities of simulations must be kept in mind. If a simulation requires a large number of computations, only a relatively small set of alternatives should be carried over. The analysis as well as the metrics dictate the way design alternatives are generated.

Generating and evaluating alternatives could be done consecutively, iteratively, or in phases. The analysis performed during these steps predicts the performance, cost, collaboration issues, etc. Decisions are made using the scores each alternative receives based on the value established at an earlier step. However, in order for this process to be executed properly, several requirements specific to design and simulations must be addressed.

After a description scheme is set, alternatives are generated. Finally, for analysis, computer simulations are used as previously alluded to. Various computer simulation types (such as discrete event, agent based, and Petri nets) are, or are at the very least becoming, industry standards. However, the means to accomplish design description and analysis are still vague. For example, one can use diagrams and pictures to represent architectures; however, there is little agreement on how those diagrams and pictures are stored or represented (especially on a computer). Both generic markup languages as well as domain specific languages are used to store architectural information in a computer readable way, and each approach has its own strengths and weaknesses [104]. Architectures will be revisited in Chapter 3 and 5 again. The following sections of this chapter provide the background to the research work as shown in Figure 4.

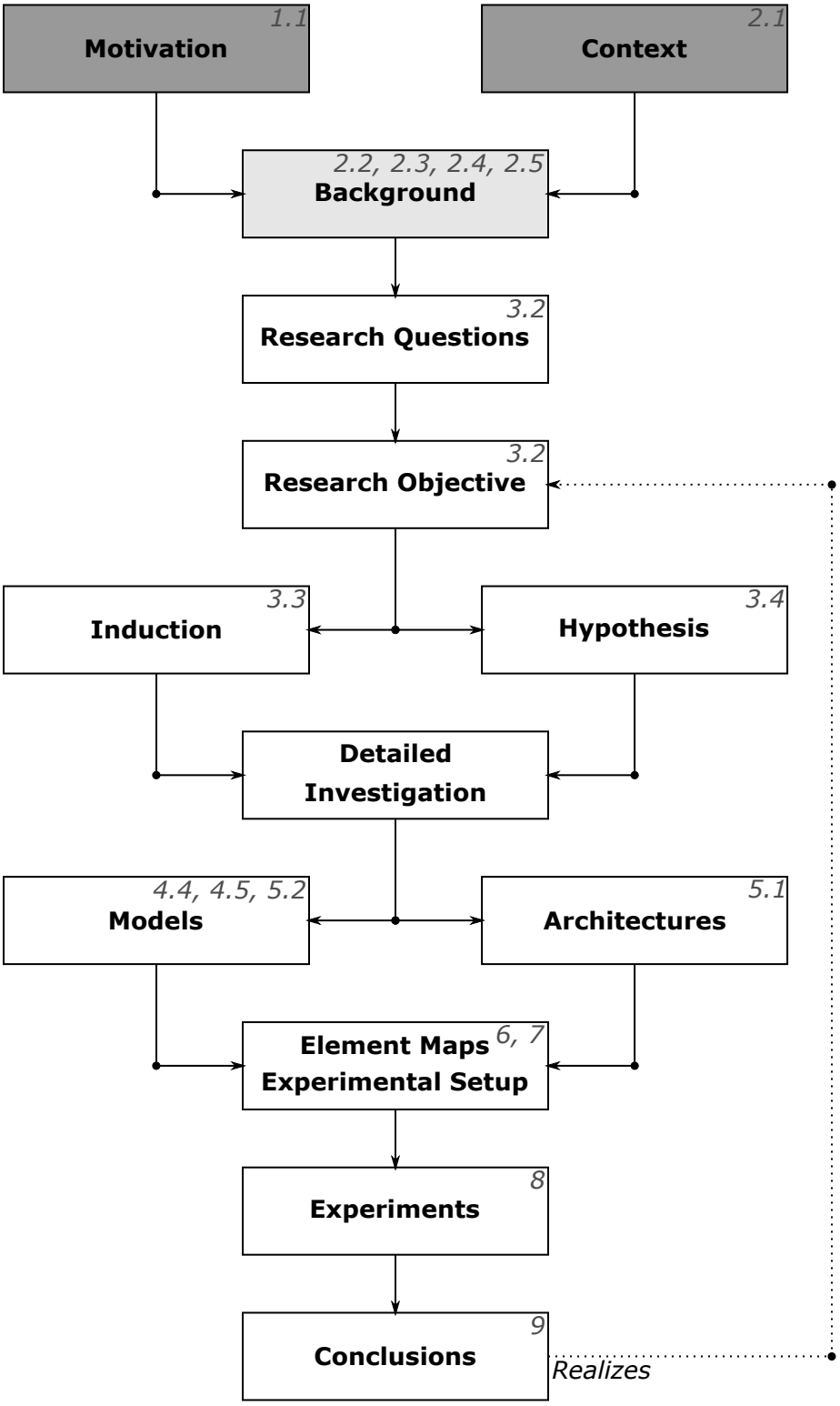


Figure 4: Context is set. The next sections provide the background to the research.

2.2 Systems

Systems are defined in various ways. The word originates from two Greek words *σύν* and *ίστηνι* which roughly translate into "something made of many existing entities". The word system is usually associated with organization and method. Below are examples from dictionaries and scientific encyclopedias:

- an assemblage of parts that form a unified whole [144], a coherent unification [86], a group of related structures [156], a set of units combined by nature or art to form an integral, organic, or organized whole [86]

- a collection of particles or interacting components considered to form a distinct physical entity for the purpose of study or identification [144], a method of organizing entities or terms; in particular, organizing such entities into a larger aggregate [156], an aggregation or assemblage of objects joined in regular interaction or interdependence [86], an orderly working totality [86]

- a combination of components, elements, subsystems, and operating procedures, functioning together to achieve some objective [144], a complex unity formed of many often diverse parts subject to a common plan or serving a common purpose [86], a combination of several pieces of equipment integrated to perform a specific function [156], a group of devices or artificial objects forming a network or used for a common purpose [86]

Based on the above dictionary definitions, it can be said that a system is defined generally by two aspects. The first is the distinction of what is internal versus what is external to it. Internal elements are called *parts*, *subsystems*, or *components* of the system. External elements are usually called *the environment*. The second aspect is the relationship between the elements. In most cases, the relationships of the internal elements are more important than the interactions between internal and external

elements. A system also has a purpose, a function, and/or an observable effect on the environment. Cell phones, airplanes, coffee makers, dishwashers, microphones, satellites, star clusters, organisms, ant nests, traffic lights, and control volumes all fit the description of a system.

The size is not a criterion for the definition of a system. The tiniest or the largest thing can be described as a system. In fact, the same physical object can be described as a system or a component depending on the purpose of the definition. For example, a desktop computer is a collection of many parts: a motherboard, a processor unit, memory modules, a hard drive, a power supply unit, an optical disk drive, and a case that holds these parts together. However, the same computer can be thought of a component when it comes to the use of it: a person needs a screen, a keyboard, and a mouse to work with a desktop computer. These peripherals and the computer make up yet another system. The particular definition of systems depends on how the definition will be used.

With such a broad definition, almost anything in the universe can be called a system, which may render the definition useless. Therefore, it is necessary to add further rules/limitations to the definition. The focus of this work will be on human-made/designed systems. The author defines human-made systems as systems that were invented, conceptualized, and/or designed by humans. The production can be automated, or even natural; however, the design and the determination of purpose must be performed by a human. From an engineering perspective this classification is important. A very widely used definition for man-made systems is given by International Organization for Standardization (ISO), International Electrotechnical Commission (IEC), Institute of Electrical and Electronics Engineers (IEEE), and International Council on Systems Engineering (INCOSE): *a system is a combination of interacting elements organized to achieve one or more stated purposes* [7, 93].

Three other notable definitions for a man-made system are as follows.

1. "an interdependent group of people, objects, and procedures constituted to achieve defined objectives or some operational role by performing specified functions" [177]

2. "a collection of interacting components organized to accomplish a specific function or set of functions within a specific environment" [181]

3. "an arrangement or grouping of objects that operate together with a common purpose" [78]

In these definitions each component is assigned one or more functions to be performed. They, therefore, assume that all parts of the system have some kind of a function and no part is irrelevant.

Designed systems are used in every aspect of our lives. From health monitoring to processing web search queries, from carpet cleaning to distributing electrical power, people depend on systems. This document was authored on a computer, made out of many smaller systems, each of which were produced by some industrial production system, shipped to assembly centers by transportation systems. Engineered systems are a ubiquitous part of our lives; therefore, from the progress-oriented perspective of an engineer, it is important to design systems with improved capabilities and efficiencies. Similarly, from a business perspective, value is created by either solving an yet unsolved problem or reducing waste in a solution.

Furthermore, this work deals with collaborative and competitive systems. Systems operating in isolation are not dealt with. Such systems have very focused and specialized use cases, and their design is much simpler and easier understood. This work will not go into designing specific isolated systems. It will focus on determining the necessary interacting systems for multiple purposes. The reader is reminded that once the necessary systems are decided upon, more isolated systems design activities can be used. In other words, if one decides to perform system of systems design on

an isolated system, the conclusion of the analysis will be that *the system is needed*.

Collaborative and competitive systems are systems that are constantly communicating with, sensing, serving, counteracting, enabling, disabling, each other. The operation of such systems not only depends on physics, but also the behaviors, performances, and past actions of interacting systems. It is, therefore, impossible to measure the value of a collaborative/competitive system outside of its *ecosystem* of interacting systems. For a fighter airplane, it may be useful to list its rate of climb, maximum speed, and armaments it can carry; however, it is more useful to define its actual influence on the many missions its operators are flying, such as mission completion time, percent targets neutralized, combat losses, and fratricide rate.

These metrics are more difficult to calculate for a systems developer, because they are not metrics for the system in isolation; they are metrics for the system being used in one or more scenarios and in the context of a group of other heterogeneous systems. The determination/measurement/estimation of such metrics require an operationally focused analysis requiring the consideration of the system's interaction with the uncertain external environment, rather than an analysis of the internal workings of the bounded system. Before such analyses can be discussed, systems of systems must be introduced.

2.3 Systems of systems

There has been an increase in interest in system of systems in the military acquisition communities, who are always looking for new capabilities since the capabilities-based assessment was accepted as a paradigm. In fact the definition that the DOD offers for a system of systems is very telling of its suitability for capability-based acquisition considerations: "*a system of systems is a set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities*" [153]. Military assets become systems of systems given their force

structure, the organizations they belong to, the means by which they are connected to each other, and the roles they are assigned to fill. Other notable definitions of systems of systems are given below.

- "A system-of-systems is an assemblage of components which individually may be regarded as systems which possesses two additional properties: operational and managerial independence of the components" [130].
- "Groups of systems, each of which individually provides its own mission capability, that can be operated collectively to achieve an independent, and usually larger, common mission capability" [155].
- "A set of different systems so connected or related as to produce results unachievable by the individual systems alone" [117].

Each of the definitions point to the well-accepted characteristics of systems of systems: multiple systems integrated into a larger system, each system is independently useful, and systems collaborate to achieve larger goals. Additionally, some of the definitions also stress independent management. The author believes that the DOD definition is precise and concise enough to accept for the purposes of this work.

The word *capability* requires some explanation. Capabilities are high-level goals in the context of a military operation. Although the DOD Dictionary of Military and Associated Terms does not specifically define it, many definitions that use the word *capability* point to a high-level goal [62]. The Joint Capabilities Integration and Development System defines capability simply as the ability to execute a specified course of action [84]. It could range from rapidly building forward operating bases to hitting a high-priority time-critical target; from keeping up with the logistics requirements to the ability to employ enough personnel. DOD defines capability as "*the ability to execute a specified course of action*" [62]. Some examples may clear any confusion. The ability to fly at supersonic speeds is a performance criterion for

an aircraft system; however, being able to intercept incoming enemy fighters so many miles ahead of the national airspace is a capability. Withstanding a mine blast is a performance criterion for an armored vehicle, but safely operating within enemy-held urban areas is a capability.

One important difference between a system and a system of systems is that the elements within a system of systems are owned and operated by different entities who are free to make their own decisions. The cooperation arises from a mutual benefit, not by placement, automation, or definition. A fighter pilot's decision to follow orders is different from an engine piston moving faster with the increased fuel flow. The former chooses to follow orders (there may be conditions in which he/she cannot), whereas the latter is physically forced to move faster. It can be said that the elements within a system of systems are more loosely connected with each other. There are certain types of systems of systems that are very limiting in the design decision phase; however, the operational actions do enjoy some freedom.

There are several types of systems of systems. The categories as defined by the DOD are *virtual*, *collaborative*, *acknowledged*, and *directed* [58, 130, 53]. These categories are becoming an accepted way of classifying systems of systems outside of the DOD as well. The classification is based on operational freedom and goes from the ultimate freedom of making design or integration decisions to complete lack thereof.

A very well-known example is the Internet. Due to its design goal of connecting computers worldwide, the Internet is a great example of a successful system of systems. The Internet is a collection of standards, procedures, and systems that adhere to those standards and procedures. The owners of these computers are incredibly mixed: personal computer users, internet service providers, governments, national and multinational entities, non-government organizations, schools, media companies that generate and distribute content, etc. Today, companies, small and large alike,

depend on this network to conduct their business. The power of this system of systems comes from its openness, little perceived effort to be part of it (i.e., connecting to it), and ease of use via graphical user interfaces (e.g., web pages, apps, games). With the vision of *the Internet of Things* well underway, the Internet is becoming an inescapable part of human life: it is in the living rooms (televisions), on the move (mobile phones), in coffee shops (wireless local area networks), or even within human bodies (connected heart monitors).

Another example is an air transportation system. These systems exist in every country, where there is an airport or airplanes fly through the airspace. As airplanes use the airports and the airspace, the air traffic controllers survey the area and deconflict paths. In order for this to work, all systems are in communication with related parties at all times. Failures are handled with extreme care and pre-determined rules and procedures. Radars, very high frequency omnidirectional radio range beacons, global positioning system satellites, aircraft, radio towers, and other connected elements are all part of air transportation systems just as airplanes, airlines, passengers and cargo, and fuel infrastructures are. This system is glued together by human operators and compared to the Internet, a smaller degree of automation is employed. This aspect shows the flexibility of systems of systems in their implementation of interfacing, decision making, and operation.

One last example considered here is a basketball team. Even if the business side of the team is ignored, the actual players and their coach form a system of systems. Each player, even though executing actions within teamwork, makes his/her own decisions, but is simply part of a grander scheme of tactics. The players are heterogeneous systems: they each assume the role of pivot, power forward, small forward, point guard, and shooting guard. Some switch between roles. All players adhere to the rules of the game and act, move, and perform accordingly (e.g., dribble not travel, handle the ball with hands not feet). Observing several games of basketball also uncovers

some emergent behavior[1] such as the last five minutes of the game lasting more than fifteen-twenty minutes due to clock stoppages and time-outs. Players do get replaced by other players during the games or between seasons by transferring, but the team still stands as a system of systems. This type of systems of systems is completely non-automated and operates gracefully through practice and training. Communication between systems is done by hand signs, shouting, or simple observation of actions.

The Internet is of the collaborative type: standards and committees exist to manage protocols but the cooperation is mostly voluntary. The air traffic control is a directed system as governments enforce it via laws in their jurisdictions, but is a collaborative one in the global scale, as governments work together under organizations such as International Civil Aviation Organization (ICAO). A basketball team is a directed system of systems because the coach tells the players to do certain moves even though there is considerable freedom in the executions of the prescribed actions.

Returning to the DOD's definition, some arrangements of systems will perform better, be more robust, and cost less given the spectrum of capabilities they are expected to deliver in various situations. The system of systems engineer's goal is finding the best solution given the set of desired capabilities. This problem is an inverse design problem, not unfamiliar to the aerospace community, where the performance metrics are known via standards, requirements, and desirements, while its design parameters are unknown. However, because analysis cannot proceed backwards (i.e., consequences do not lead to the causes), the analysis is done by seeding the problem with some possible designs and observing their performances. Later the problem is inverted mathematically or via trial-and-error to obtain the required design parameters from the given requirements.

[1]Emergent behaviors are patterns that come to exist without any intervention to make them exist. Such behaviors are usually the result of the arrangement of systems or the rules that apply to them. Because they are previously unplanned for, they are named emergent. Emergent behavior will be more thoroughly investigated later.

Furthermore, the prospective systems to be included in the system of systems may already be in operation, development, or consideration. As discussed earlier in Chapter 1, systems of systems are not usually designed from scratch; a part of them already exist and is in operation, while another part is in design [52]. The consequence is that there is considerable mismatch in data availability between the existing and non-existing components. In addition to this mismatch, the available data for the non-existing systems is likely to be uncertain in nature. This uncertainty is due to the lack of knowledge about the system and is, therefore, of the epistemic type [106]. Epistemic uncertainty is sometimes referred to as type B uncertainty or uncertainty due to lack of knowledge. Epistemic uncertainty can be reduced in time with better analysis on the systems that make up the system of systems [106]. Reducing epistemic uncertainty requires time, resources, and effort.

There may be uncertainty in the performance of existing systems as well. For example, a fighter jet may take two minutes to find a target or it may loiter for hours to find it. Individual mission performance of aircraft will vary from time to time as they may be in different phases of their maintenance schedules, different pilots may be flying them, or different weather conditions may exist in the area of operations. The amount of uncertainty in the design is disconcerting and this aspect of system of systems cannot be downplayed or ignored. However, determination of all performance metrics for every conceivable scenario is an arduous task and should be scoped out of modeling for practical purposes. The variability of performance should be included in the analyses as uncertainty, limiting depth and fidelity of models. Therefore, this uncertainty is of the aleatory type [106]. Aleatory uncertainties are sometimes referred to as type A uncertainties or uncertainty due to variability. Aleatory uncertainties are not reduced with more analysis, as they are inherent to the nature of systems that make up the system of systems [106]. It is important to separate epistemic and aleatory uncertainty in risk analysis [100].

With their characteristics discussed above, systems of systems are documented in design descriptions called architectures. These descriptions are designed to handle the varying operational freedom, unending development cycles, multiple sources and types of uncertainties, and the management of multiple capabilities and programs. A discussion of what architectures are and how they are used is given next.

2.4 Architectures

As discussed before, design is partially a decision making process [94, 179]. Design decisions made throughout the design process result in a final design; therefore, the final design can be thought of the aggregation of all design decisions. These attributes of the design are encoded in a design description, which is usually in the form of an *associative array*. This construct is also known as a *map*, a *symbol table*, or a *dictionary* [137]. An associative array is made out of key-value pairs. Each value has an associated key, which is assumed to exist and is unique. In the context of design, the variables are stored as keys and their settings/values are stored as values. Figure 5 depicts an example associative array.

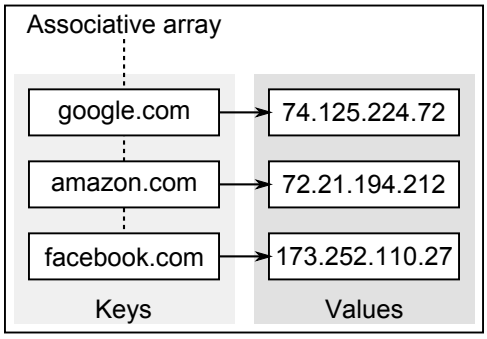


Figure 5: Associative arrays store values linked to corresponding keys

Design descriptions must have two fundamental characteristics. Firstly, one alternative shall not have two different but valid descriptions for the same purpose. In other words, design descriptions are unique for alternatives. This requirement will ensure that alternatives are unique and that the same alternative is not analyzed

twice. Secondly, it distinguishes one alternative from another. In other words, two alternatives shall not have the same design description, unless they are the same alternative or equivalent alternatives under the analysis being performed. This requirement ensures that two distinct designs are not considered to be the same design, and hence to have the same performance during the analysis step. Figure 6 shows this relationship graphically.

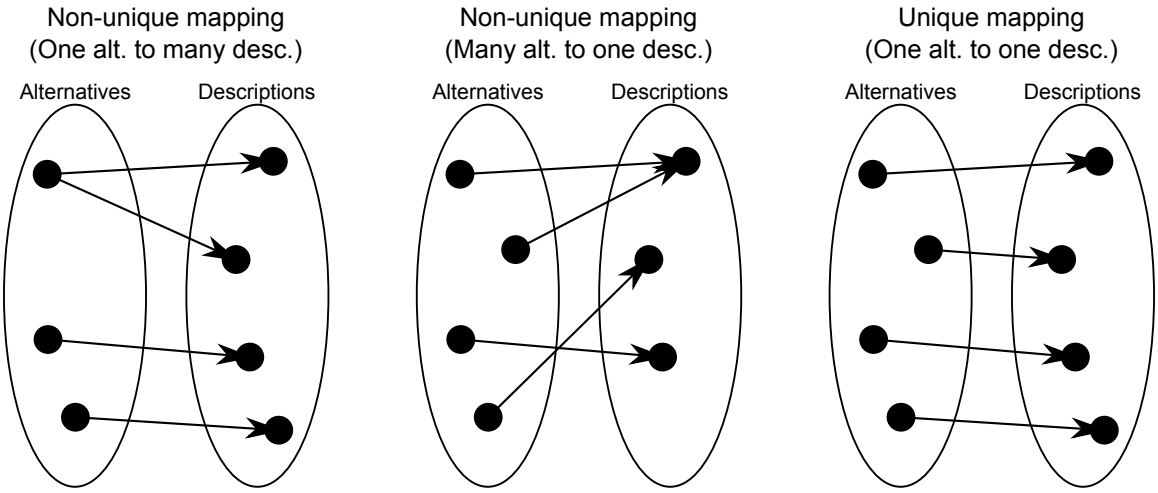


Figure 6: Mapping between the set of designs and the set of representational models

In engineering design, one of the steps is the description of design alternatives. System of systems design is no different in this aspect. A design engineer needs a way of describing a prospective system of systems design alternative before it can be analyzed. Certain design attributes relate to the conceptual properties of the design, while others relate to specific settings of a given property. For example, a flying system can be a fixed wing aircraft, an airship, or a rotary wing aircraft; the implementation is a conceptual property of a design. Conversely, the span of a wing being 93 feet long is not a conceptual property; it is a setting of a property.

Conceptual properties open and close entire spaces inside design spaces. For example, the decision to go forward with a rotary wing aircraft eliminates design variables such as wing span, flap sizes, and engine pylon placements, but the decision opens a design space spanned by design variables such as disk area, number of rotor

blades, or landing skid length. Inside the associative array, these alternatives can be represented as yet another associative array (nested).

Without a good taxonomy of possible elements, design descriptions are not a good way of documenting what the design is. Even with a good taxonomy, they may be very impractical in describing the entirety of a complex system. For example, in a turbofan engine, the bleed air ratio by itself is not enough to perform many analyses such as calculating turbine blade temperatures. For such an analysis, the interaction between the compressor and the turbine must be known. Because system elements are always interacting with each other, the interactions must be more clearly defined for an effective description of a system. Such design descriptions are close to what are widely known as *representational models* in science and engineering. In systems engineering, they can also be referred to as *system architectures*; a term defined in many ways by different entities:

- "The organizational structure of a system or component" [1]
- "System, product, or service architectures depict the summation of a system's entities and capabilities levels of abstraction that support all phases of deployment, operations, and support" [193]
- "Fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution" [7]

Design alternatives must first be described, next they must be analyzed, and lastly design decisions must be made in any design process. Analysis cannot be performed prior to description and the description is usually taken for granted for most engineering design activities. For example, in fixed wing aircraft design, the description of a candidate is represented as a list of attributes (e.g., T_{SL}/W_{TO}, W_{TO}/S, $\mathcal{AR}$). Some of these description elements may be changing throughout the flight (i.e.,

updated via mission simulation), whereas others may be fixed. A similar description scheme of a system of systems design candidate must be established.

System architectures then hold more information than regular system descriptions because they are standardized and include information other than a physical taxonomy/decomposition. Until recently, however, system architectures have been mainly used for documentation purposes. Static, hard- or soft-copy documents have been used as data containers for system architectures. Such containers include technical reports, presentation slides, manuals, doctrine documents, graphs and charts, etc. However, the opinion on how to use architectures is rapidly changing [204, 90, 157, 80]. If such documents were made in a computer readable fashion, models created to analyze designs can use them as inputs in support of the simulation. This is the main idea behind the DOD's push for *executable architectures*. Department of Defense Architecture Framework (DoDAF) is a good framework for developing system architectures and is slowly evolving to include features that help with analysis steps [143]. In this context, executable architectures provide the means to conduct dynamic analysis of systems, and are emerging as a supporting method [202]. The DoDAF document also hints further development on the modeling and simulation and consistency/logic checks that integrate seamlessly with DoDAF[59].

It is safe to assume that systems of systems can be described by architectures and that some form of simulation will be used to analyze them. In turn, the outputs of these simulations will be used to assess potential system of systems designs or upgrades, as part of IPPD's step 6. These assumptions are justifiable by government mandates, recommended practices, industry trends, and practical considerations [195, 153, 58]. This research focuses on the descriptions of systems of systems and their coupled nature to the system of systems analysis using computer models.

Ideally, the description of a system should be independent of its analysis so that it is only influenced by what is known about the design not dictated by the analysis

requirements. The independence is desired because the description should be about the system, not its analysis. The description must keep all potentially relevant aspects of a system and store them in a useful form. However, because the purpose of description is analysis, the relevant aspects are, more often than not, related to analysis methods. Several examples will be considered below.

The free body diagram is a well-known system description method in the field of mechanics. An example is given in Figure 7. The free body diagram is a way of describing a system so that its dynamics could be analyzed. The systems it describes fit the description given in Section 2.2.

- It has internal elements (bars, strings, point masses, etc.),
- There are relationships between internal elements (joints),
- It has external elements (accelerations such as gravity, curved paths, forces, etc.),
- It has relationships with the external elements (Newton's Laws and boundary conditions),
- The mechanism itself has a purpose/function,
- The mechanism influences and is influenced by the environment during its operation.

The free body diagram breaks a system into multiple *view*s. Each rigid body and massless component gets its own view. The free body diagrams have all the information needed to model the system dynamically using Euler's First and Second Laws: the linear and angular momentum of bodies with mass and inertia is increased proportional to the sum of the forces and moments acting on them [56]. Also, massless bodies have to be in static equilibrium. However, if one was to use d'Alembert's

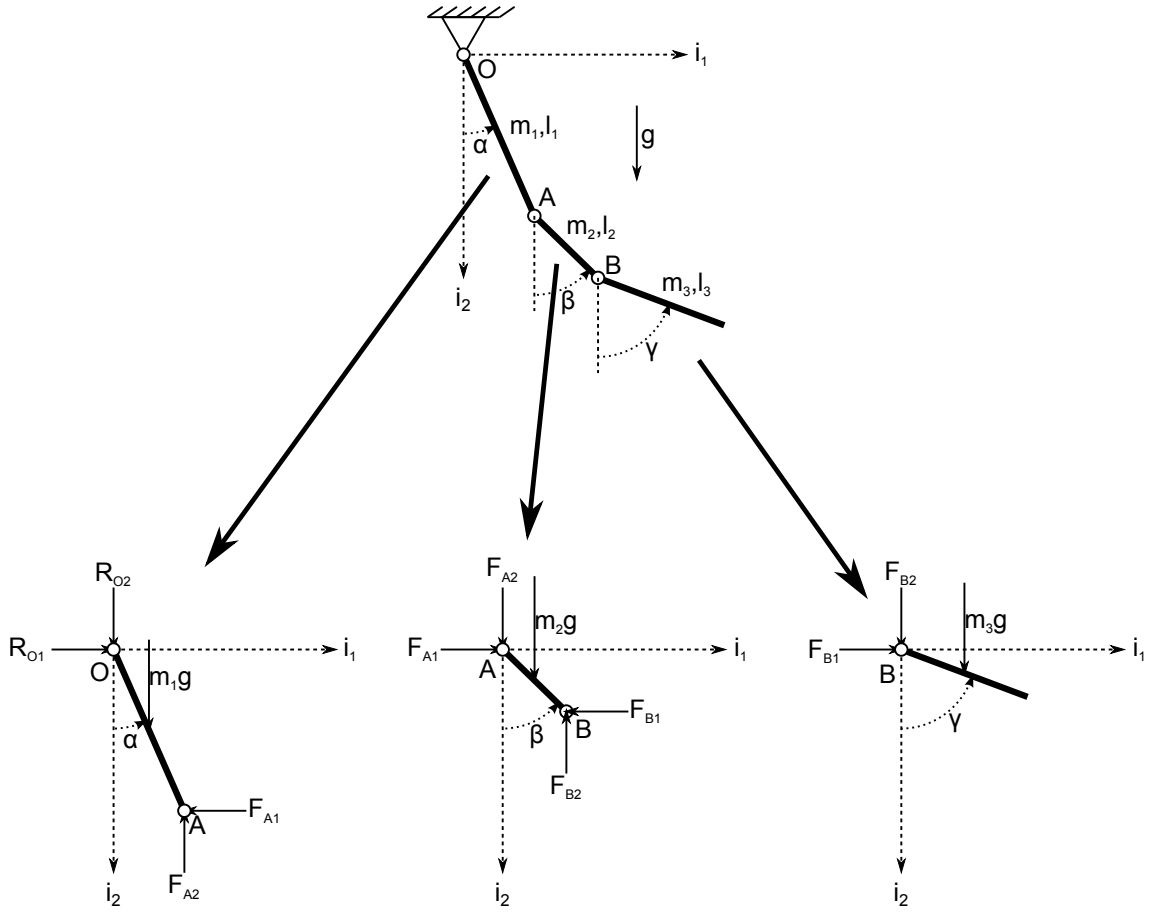


Figure 7: Free body diagrams decompose the problem into more focused views

Principle to model the system, there is some extra and some missing information from the free body diagrams. d'Alembert's Principle does not require the knowledge of reaction or internal forces, but requires the knowledge of kinematically admissible virtual displacements [27]. It is clear that, at least in this case, system description has some relationship with system analysis.

Another example is from the field of mathematics. A system of equations with shared variables can be represented in a matrix form. By convention, the variables are represented by the columns of the matrix and the equations are represented by the rows of the matrix. An example is given in Equations 1–4. Using linear algebraic methods such as Gaussian elimination, the system of equations can now be solved. The matrix representation of a system of equations is an architectural view of it. It

takes advantage of the order in the equations and compresses them to a structure that holds only the information needed for the analysis to be performed. Once the matrix representation is used, the information on the variables is lost. It has to be kept separately until the analysis is complete. The matrix representation of a system of equations is another example of a description being closely tied to the analysis of system.

$$3x_1 + 4x_2 + 5x_3 = 15 \tag{1}$$

$$5x_2 + 3x_3 = 10 \tag{2}$$

$$2x_1 + 10x_2 - x_3 = -9 \tag{3}$$

$$\left[\begin{array}{ccc|c} 3 & 4 & 2 & 15 \\ 0 & 5 & 3 & 10 \\ 2 & 10 & -1 & -9 \end{array}\right] \tag{4}$$

As seen in the presented examples, the description is usually linked to the analysis. In fact, this is done out of necessity: the description of the system is a minimal representation of the system with some purpose—in this case: analysis—in mind. Without the knowledge of what will be done with the representation, nothing can be ignored, scoped out, or simplified, because there is no prior knowledge of what is needed later for analysis. This contradicts the purpose of design description: capturing every known detail about the design so that any desired analysis can be performed later. It is not specific to a single analysis method.

Now that the need for and the requirements of system of systems design descriptions have been outlined, a description scheme must be picked. The description that is sought will describe systems of systems which were defined earlier as a set or arrangement of systems that results from independent systems integrated into a larger system that delivers unique capabilities [153]. One solution stands out in the DOD's dictionary: a *technical architecture* is defined as a minimal set of rules governing the

arrangement, interaction, and interdependence of the *parts or elements* whose purpose is to ensure that a conformant *system* satisfies a specified set of *requirements* [62]. The similarities between the two definitions are striking enough that it becomes natural to define technical architectures as descriptions of a systems of systems. In the rest of the document, technical architectures will be referred to as architectures for brevity.

The word minimal in the definition of an architecture requires some elaboration. An architecture is only a *minimal* description of a system and as such, it does not contain every detail about the system. Architectures omit certain details for the sake of brevity, clarity of the big picture, and possibly uncertainty. This simplification will prove to be useful later; however, the design engineer must make sure that the description is detailed enough to distinguish between two distinct design alternatives.

2.5 Modeling and simulation

After the description, the next step in design is analysis. Analysis is a way of evaluating prospective design alternatives to find differences in their performances. It is this difference in performance that makes one design more or less attractive compared to another design. The basis of all design decisions is the results of the analysis step.

Analysis can be done in many ways such as quantitative vs. qualitative, exact vs. approximate, numerical vs. analytic. Because the number of system of systems design alternatives is usually very large, it is important that their analyses be done on a computer in an automated fashion. Also, ultimately, many alternatives need to be compared against each other. Results that are quantitative also help with making clearer and more definitive comparisons. One obvious option to obtain such results is the use of computer simulations.

Computer models are the modeler's knowledge and understanding about the real

system expressed in a computer-executable format. A discussion on models and computer models is presented in Chapters 4 and 5. For the design engineer, however, the analysis can be a black box that will be called whenever a design is to be evaluated. This function takes in the design description, or part of it, and returns the performance of that design. By definition, this is a map from one space to another, as depicted in Figure 8.

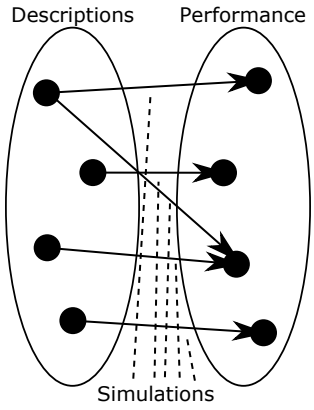


Figure 8: Simulations map a system's description to its predicted performance

However, in all engineering cases, the design engineer has considerable knowledge in the analysis calculations. Therefore, there are compelling reasons for the analysis to be a transparent model as opposed to a black box. For example, an aerospace engineer understands that increasing the aspect ratio of a wing will decrease its induced drag coefficient. If the engineers can see how the model uses the aspect ratio input, they can increase their trust and confidence in the model and consequently the outputs it is generating. Even better, the engineers can come up with higher quality experiments, modify internals as they see fit, and create a more suitable analysis package for the design at hand.

So far, a single modeling environment to perform all system of systems analyses has not emerged. This is likely due to reasons such as the diversity of the modeling types that can be used to model systems of systems. For example, agent-based modeling—a suitable modeling type for system of systems simulation—may perform very well in

the analysis of some systems of systems; however, there is no guarantee that it will perform well in the analysis of other systems of systems. In fact, the literature on system of systems modeling has examples of the following modeling types:

- Graph theory [64, 91, 24, 88, 50, 75, 115, 77]
- Probability theory (Prob) [175, 77]
- System dynamics (SD) [29, 44, 43, 88]
- Markov chains (MC) [50, 88, 17, 131]
- Petri nets (PN) [75, 88, 172, 109, 79, 124]
- Queueing theory (QT) [17, 65]
- Discrete event (DES) [88, 102, 20, 69, 37]
- Agent-based (ABM) [24, 77, 172, 99, 204]

Each of these modeling types will be discussed in Chapter 5. However, it is an interesting fact that so many modeling types are suitable for system of systems modeling and that there has been no practical guidance on how to select the modeling type based on the problem at hand. The focus point of this research targets these peculiarities. The questions to ponder are presented below.

- Why are so many modeling types used for system of systems problems?
- Is there a modeling type that outperforms others?
- When should one modeling type be used instead of the others?
- Do the descriptions affect what can/cannot be modeled?

A crucial step in computer modeling is the formulation of a conceptual model. In many cases the understanding of the real system/world cannot be directly converted into machine code. A major part of the modeling actually happens when the scientist/engineer tries to conceptualize in their minds how a system/nature works. This conceptualization, when transferred to a medium in a syntactically consistent way, is called a *conceptual model*. In many cases, conceptual models are graphical expressions of the understanding of the modeler. In other words, they are the mechanical reasons behind observed phenomena, and as such very closely related to hypotheses in science. They have the power to explain observations and make predictions.

The next chapter seeks answers to these questions by stating them in the form of hypotheses. Each hypothesis will also include observations that led to its formulation, as well as planned experiments to assess its validity.

CHAPTER III

RESEARCH ARGUMENTS AND WORK

> When a man desires ardently to know the truth, his first effort will be to imagine what that truth can be.
>
> Charles Sanders Peirce [158]

The research work outlined herein is aimed at guiding an engineer through modeling (i.e., creation of an environment) for the design of systems of systems. It has been established that modeling—especially computer modeling—is an appropriate analysis approach for system of systems design. There are significant challenges in simulating systems of systems. This chapter summarizes some of these difficulties as observations, tries to explain them using hypotheses, inductions, and the corresponding experiments that will be used test them, and finally outlines a technical approach to solve some of the system of systems simulation challenges. Before proceeding with the research topic, expectations and research formulation process are set for the reader. This work includes a main induction and a main hypothesis. Their tests involve different mechanism. What follows is a discussion on these two types of arguments.

3.1 Gaining knowledge via synthetic arguments

Arguments come in various types, both content-wise and form-wise. There have been many attempts at creating a taxonomy of arguments since the topic was initiated by Aristotle. Aristotle believed that the only valid argument is an *analytic* one; a type of argument which guarantees its result from the given premises. Other names for analytic arguments are deductive, formal, and explicative. As long as the premises

are true, the consequent will also be true. Using analytic argumentation, one can infer consequences *a priori*, i.e., without having to execute them. Therefore, analytic reasoning is closely related with forecasting and prediction. However, if all arguments were deductive, it would mean that argumentation is merely rearranging information to make implicit information explicit. The other kind of arguments are called *synthetic*; a type of argument that produces new information not previously included in the premises but hinted at its existence. Synthetic (or ampliative) arguments are the foundation of empirical sciences. They can only be generated *a posteriori*, meaning that observations and experiments must be carried out before knowledge is gained.

Synthetic arguments also have sub-types. The more commonly accepted types are called inductions and abductions. Abductions are more commonly known under the name of hypotheses. Unfortunately, the word hypothesis has been overextended to mean other concepts that are not arguments. For example, in the common language, *hypothetically speaking* implies an unreal situation. However, hypotheses in science are guesses that the scientist has a reason to believe and tries very hard to disprove. Hypotheses that survive this intense scrutiny are taken as scientific facts (they never cease to be open for disproving, however). Similar to hypotheses, inductions are falsifiable by empiricism. Inductions are mainly generalizations from individual cases to trends. If certain cases do not follow the same trend, inductions can be invalidated.

Peirce was the first philosopher to distinguish inductions from hypotheses. His work led to useful discussion on how to test hypotheses and inductions. The different nature between the two inferences requires different approaches to validate them. For hypotheses, a scientist devises a taxing experiment to falsify the claim. Over time, as such experiments fail to invalidate the hypothesis, it gets accepted as a trusted fact. Hypotheses cannot be tested directly, they involve careful development of *consequences if true* and compared with new observations. Inductions, however, are only adjusted (if possible) as new cases are tried. Inductions are closely related

to natural laws as hypotheses are closely related to natural mechanisms.

3.2 Motivating characteristics of the problem

As discussed in Chapter 2, system of systems modeling and simulation efforts have been somewhat ad hoc. The lack of guidance for simulation is mainly due to the fact that system of systems are very flexible in their nature. They can be space systems thousands of miles apart, small ant colonies fitting within a cubic meter, or virtual systems that only exist on computer hardware and not in the physical domain. This is the same reason why the guidance on modeling has been lacking.

There are examples of modeling similar systems of systems using varying modeling approaches such as discrete event simulations, graphs, or agent-based models with Petri nets [102, 77, 172]. None of these choices are right or wrong; however, the reasons behind why one was selected over the other are not always clear. There is no overall guidance on how to select modeling types for systems of systems problems. This is the single most important motivation for the research work outlined in this chapter: help modelers make decisions on which modeling type to employ.

The selection can occur due to the modeler's expertise in one modeling type or an aversion against trying new modeling types in a form of neophobia or fear of the unknown. This is similar to a exploration-vs-exploitation problem commonly encountered in engineering design or optimization. When a solution space is not explored during early stages, the exploitation approach tends to settle in a local optimum. Conversely, if exploitation is not employed soon enough, the alternatives never converge to an optimum solution. There is a balance between learning about different modeling types and creating efficient models.

The problem of not knowing which model type may best serve the modeler's interest as well as limited knowledge about each model mirrors a class of problems called *bandit problems*. The main idea of a bandit problem is that a decision maker

has a limited amount of resources to distribute among a multitude of options with unknown return [31]. As resources are spent on one option, knowledge about it is gained. The decision making process is turned into a sequential exploration of alternatives and once an option is believed to be the best, spending the rest of the resources on that option. The aim of this work is to help the modeler gain some knowledge about the return of each modeling type, before significant effort is spent in the actual modeling. It is meant to make the starting point for the bandit problem less uncertain.

From this point on, it will be assumed that the design activity will require a computer model and focus will be on how to perform this modeling step of the system of systems design. Specifically, the reader is presented with a research work on *how to choose a fitting modeling paradigm* for various types of system of systems design problems.

How can there be any guidance on modeling activities on such a spectrum of engineered systems? If one focuses on every possible product family individually, a general guidance cannot be obtained. Therefore, system of systems modeling must be separated from modeling specific systems. Guided by the definitions of systems of systems, their modeling efforts should rather focus on the arrangement, communication, and collaboration of the constituent systems. That means focusing on what each system adds to the whole, not how it internally works or individually achieves this effect. For example, as part of a combat system of systems, an unmanned aerial vehicle (UAV) might be flying over enemy territory and providing bird's-eye view of enemy activity. In this example, the focus of the modeling should not be on how the UAV flies, avoids detection, or defends itself against enemy attacks, but on the quality of the image received, the delay of the information, and dependability of the information flow.

The focus of the modeling then begs the question: which modeling types are

suitable for system of systems modeling? Before this question can be answered, a discussion on the definition of a model is necessary. The reader is referred to Chapter 4 for the philosophical discussion of models: what they are, why they exist, and where they fit in the scientific process. Following that discussion Chapter 5 introduces a handful of modeling paradigms suited to system of systems computer simulation.

Because a number different modeling paradigms exist for system of systems simulation, a way of picking the most suitable (or a suitable set) for a problem at hand is useful. Even though computer simulations are recommended for system of systems analysis, a general guidance on how to select a suitable modeling paradigm has been lacking in the literature. However, once the model type is picked, the rest of the modeling work will be much better scoped and defined; therefore, the work will proceed much faster. One can liken this process of selecting the modeling paradigm to selecting a concept in system design.

If a less-than-ideal concept is selected in the first step, it is very difficult to fix that mistake at the later stages of systems engineering. Selecting a modeling paradigm based on prior knowledge and practicality can cause a non-ideal modeling solution just as selecting a product concept based on convenience can lead to an inferior result [67]. Figure 9 shows the increase of cost committed, cost incurred, level of knowledge, and ease of change for a engineering product across various design phases [68]. Because system design and analysis for design happens early in the acquisition phase, choosing the correct computer model to develop is expected to be important as well. Selecting a modeling type without much thought can be very costly as the mismatch in the information it provides and the information required for design can only be corrected with changing modeling types. Such a mistake is expected to reset the modeling step of design almost entirely.

Because a large part of computer modeling is software development, the difficulty of fixing modeling errors is expected to be similar to that in software development.

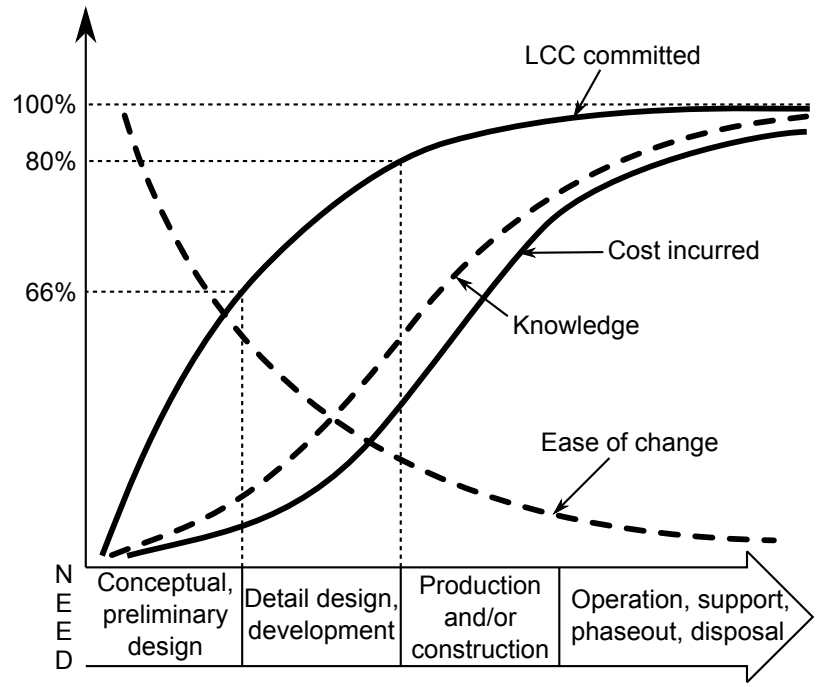


Figure 9: Ease of engineering changes decreases rapidly during conceptual design [68] (Reproduced)

If one were to discover that mistakes were made in a software project, rectifying the software development can be costly, as shown in Figure 10 [34]. Boehm argues that the larger the software project, the greater the multiplier representing the increase in effort to fix [34].

These two charts hint at the importance of making correct early decisions for modeling. The type of model selected early in the modeling process has a significant impact on the committed modeling effort and will be difficult to be swapped out for another type of model. Therefore, selecting the correct modeling type early in the design process is critical. Modeling can be seen as a design problem within a design problem.

The importance of identifying and using the correct modeling type leads to the general research objective for the work previously given at the end of Chapter 1. The objective is repeated here to guide the subsequent discussions.

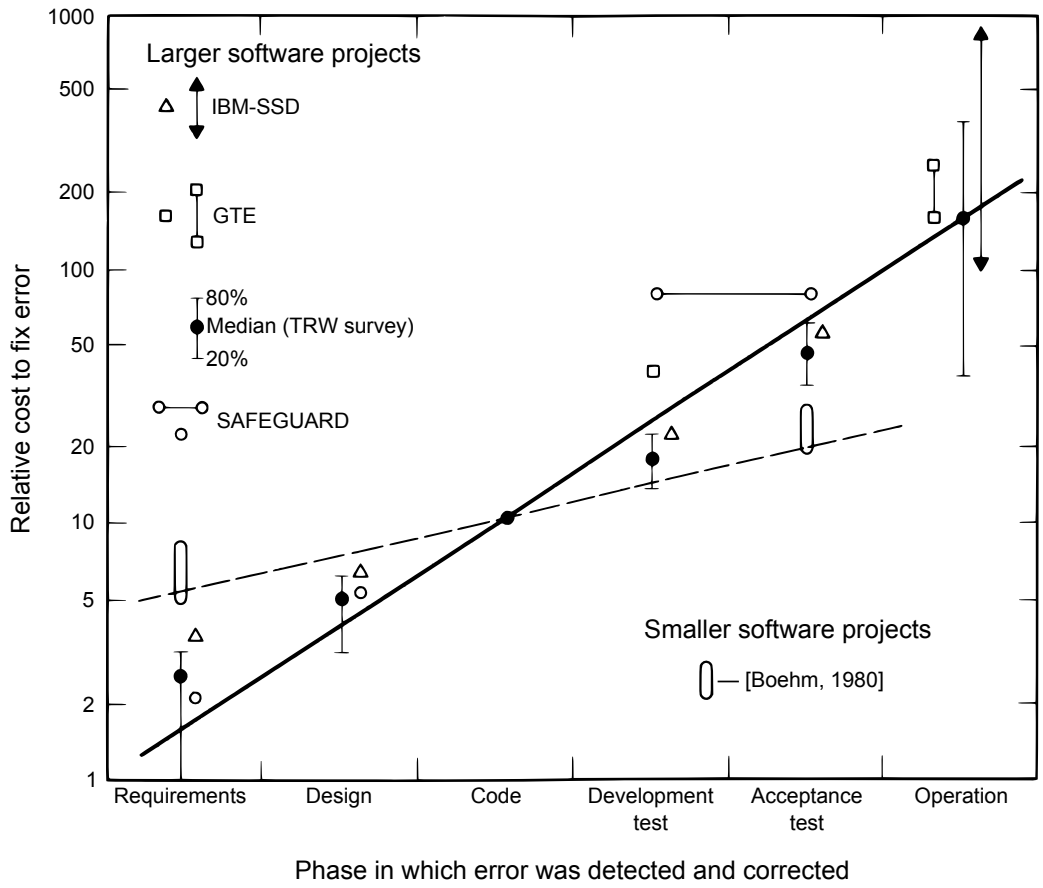


Figure 10: The effort required to make changes to software grows rapidly throughout the life-cycle [34] (Reproduced)

Research Objective. Develop a methodical approach for selecting a modeling type for analysis of systems of systems, that is useful for early design phases as well as for continuous cycles of system of systems evolution.

For the work to be considered complete, the following criteria will be used:

- The method must be extensible to new modeling types
- The execution of the method must be traceable
- The method must clearly identify the strengths as well as the weaknesses of the selected modeling type(s)
- The method must be problem-agnostic; it must be usable for a wide variety of

system of systems problems

The method is expected to be used by a knowledgeable systems engineer who has some understanding of each modeling type but not necessarily experience in modeling with them. It is not meant to separate the engineers from the decision making process; only to help them. As such, the approach is a guiding process flow that does not pick the right modeling methods and make all the decisions automatically. It offers a step-by-step process to the system modeler to pick a modeling type based on the type of architecture information they possess.

What follows is a list of aspects of the system of systems that are expected to be influential in the modeling step as shown in Figure 11. The work will investigate the effects of these expected influences and try to use them as a guide to arrive at the modeling type that fits the problem at hand. The list is organized into sections that lead to each hypothesis to be tested.

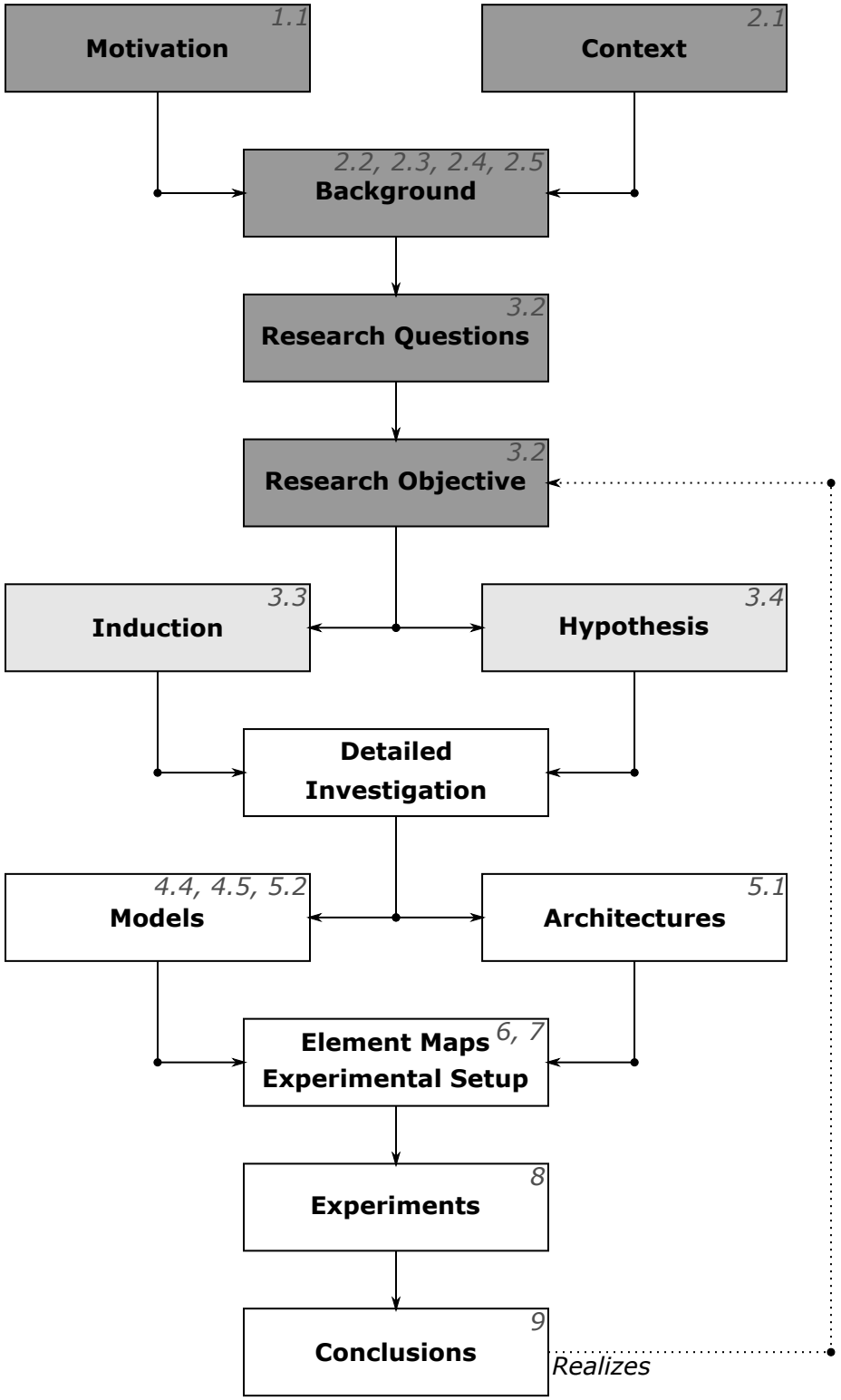


Figure 11: Research goals are set. The following is the formal development of the research arguments.

3.3 Induction: Architectures as conceptual model

As preluded to in Chapter 2, some form of an architecture framework is likely to be used during a system of systems design. This thesis will make use of DoDAF; however, the methods can be applied to any system of system architecture framework. DoDAF is used fairly widely [153, 58] and is complete enough [59] for the author's purposes to demonstrate the method.

System architectures were previously defined as "a minimal set of rules governing the arrangement, interaction, and interdependence of the parts or elements whose purpose is to ensure that a conformant system satisfies a specified set of requirements" [62]. For analysis purposes, requirements do not play a role; i.e., a system that does not meet the requirements can still be run through the analysis and assigned a performance metric. Removing the requirement portion from the definition leaves only the description of how systems are operating in collaboration with each other. Therefore, the relevant parts of system of systems architectures for this work are re-defined as follows.

System of systems architectures are a collection of views that depict an order and a modus operandi for the constituent systems as well as the interactions between them.

Independently, conceptual modeling was introduced as an early step in modeling. conceptual models are the modeler's understanding of how the world works translated into a formal language (graphical, text-based, or a combination of both). If the world is limited to a system of systems operating in a cyber-physical environment, then the relevant parts of it are the systems included in it, how those systems work internally, and how each system interacts with others.

It has to be noted that architecture viewpoints have also been used in modeling efforts in the past, which led to the suspicion that architectures and conceptual models

can be related to each other. There seems to be a great deal of similarity between an architecture and a conceptual model. In fact, they both look at the real system in question and try to describe it in a standard language. Because several of their properties match, it is induced that they are one and the same thing. The logical progression of this induction is presented below.

> system of systems architectures (A) are a collection of views[1] that depict an order (Y') and a modus operandi (Y'') for the constituent systems as well as the interactions (Y''') between them.
>
> conceptual models (B) are the modeler's understanding of the systems included (Y') in a system of systems, how those systems work (Y'') individually, and how the systems interact (Y''').
>
> $\therefore$ (by induction) System of systems architectures (A) are conceptual models (B).

This is clearly an induction as it proceeds from a case and result to a rule, i.e, a generalization. If the induction was accepted as truth, a purely deductive syllogism—it is of type Barbara[2]—would result as shown below.

Rule System of systems architectures (A) are conceptual models (B).

Case conceptual models (B) are the modeler's understanding of the systems included (Y') in a system of systems, how those systems work (Y'') individually, and how the systems interact (Y''').

Result $\therefore$ system of systems architectures (A) are a collection of views that depict an order (Y') and a modus operandi (Y'') for the constituent systems as well as the interactions (Y''') between them.

[1] Because these views are generated by the engineers, they are their understanding of how the actual system is structured or works.

[2] MaP & SaM $\rightarrow$ SaP

3.3.1 Experiment 1

The first experiment to be performed for this work is expected to increase confidence in this induction. The investigation is aimed at bridging the gap between architecture views and computer modeling. Specifically, the architecture views that are suited for model development will be identified. Additionally, which architecture views are conducive to which modeling type will be discovered. In order to find these connections, the induction must be tested with a few examples.

Inductions can be tested directly by performing more observations or studying more examples [159, 160]. It is a process of random sampling. From the observations, one can either continue to accept the induction as is or make necessary adjustments to it. Therefore, the experiments to *confirm* this induction will be to offer specific examples of system of systems architectures that are/can be converted to conceptual models. The examples that will be worked on will include:

- 2012–2013 Real World Design Challenge (RWDC) State Aviation Problem [51]
- 2011 National Airspace System Enterprise Architecture Framework (NASEAF) [19]
  - As-is architecture
  - Near-term architecture
  - Far-term architecture

These examples were chosen because the author was not involved in their development; therefore, their sampling is fair. The author could not have influenced the architectures to be more suitable for conceptual modeling. Additionally, the list chosen includes civilian and policing examples, small scale and large scale system of systems, cases with a single stakeholder making decisions and multiples. The test plan is as follows.

1. Synthesize architectural information into modeling-related chunks
2. For each modeling type identified earlier, translate architectures into conceptual models
3. Comment on whether information is missing from the architecture that is important for conceptual modeling
4. If there is modeling information missing, judge whether that information should be included in the architecture for design purposes or that modeling type is not a suitable type for early system of systems design

For this experiment, DoDAF viewpoints will be used. The focus will be on the standard views as opposed to tailored custom viewpoints that can be shipped with a DoDAF architecture. Such custom views are called fit for purpose viewpoints. The study of such views is not fruitful in this research because the applicability of results will be limited to specific examples and not be generalizable.

DoDAF architectures also usually come with companion documents that set a context, provide additional detail, or even point to places where extra information can be gathered. Such documents can be very useful for modeling; however, the method would again not be generalizable if it included the study of such companion documents. Due to these reasons, the research work sticks to the standard views.

Additionally, if an architecture can be turned into at least one conceptual model, the induction will not be modified. Only if the architecture cannot be turned into any conceptual model will the induction need to be modified. The goal of the experiment is to find negative results that disprove the induction via a counterexample.

The author expects a majority of the architecture viewpoints to be simulatable by themselves. Creating some conceptual models may require several viewpoints, which will not be counted as a failure. Many DoDAF views have similar composition to the more traditional System Modeling Language (SysML) and consequently Unified

Modeling Language (UML). These languages were constructed by borrowing graphical language elements from several conceptual models. Therefore, the expectation that some views are by themselves simulatable is not inconceivable.

A discussion on why DoDAF was selected is warranted here. There has been a large effort in developing simulation models from SysML in the last few years especially because it is based on UML[182]. However, SysML is not a perfect fit for system of systems problems. While there are attempts to make them match (e.g., capabilities modeled as requirements in SysML), the purposes of the two frameworks do not match.

DoDAF's goal is to support the DOD's acquisition process: Joint Capabilities Integration and Development System (JCIDS). The system of systems approach is highly appropriate for the Doctrine, Organization, Training, Materiel, Leadership, Personnel, Facilities (DOTMLPF [62]) as it tries to find non-materiel solutions to identified capability gaps where operational and service views are useful.

The system of systems engineer working with DoDAF might be trying to identify missing systems from the system of systems, and then define those systems using a more appropriate language such as SysML. There is a large overlap between the system views and the SysML models and software solutions exist to help architects use these standards in an interlinked way [152]. Solutions such as these require the use of DoDAF in a specific UML profile.

Ultimately, DoDAF architectures can be conveniently created in formats that are conducive for executable architecting. However, access to such architectures is limited and static schematics are used in this work. As the example base grows and the field reaches a standard practice, similar studies can be performed using DoDAF with a UML profile.

There are three separate contributions from this induction test. First, if it holds, system of systems engineers can use architectures for modeling decisions. These

architectures can even be translated into conceptual models in a semi-automated fashion. Because conceptual models can be turned into computer models very easily, architecture development for system of systems engineering will prove to be very useful in the actual analysis of design alternatives.

The second contribution is the discovery of architecture viewpoints that are especially well-suited for system of systems modeling. Not all viewpoints must be developed in an architecture; however, if modeling-oriented viewpoints do exist, the author will make recommendations to develop those viewpoints. Additionally, a table that has viewpoints as rows and modeling types as columns will be constructed. If a viewpoint is useful/necessary for a modeling type, the corresponding place in the table will be filled.

The third contribution will be in the area of improving system of systems architectures. The discoveries of the information that architectures lack to enable modeling for design will be used to recommend changes or additions to system of systems architectures. Such recommendations can further the push for executable architecting.

3.4 Hypothesis: A multitude of modeling techniques are needed

A number of modeling types have been used for simulating systems of systems in the past. A representative list was given in the last pages of Chapter 2. None of these types have been able to dominate the system of systems simulation efforts. In Chapter 2, it was mentioned that systems of systems are very much diverse and this diversity is the main reason behind an not-yet-unified modeling in the field. This claim remains to be tested.

One obvious experiment to demonstrate this using formal logic (i.e., deduction) is to construct a counterexample to the statement "there is a modeling type that can capture all aspects of all systems of systems". If a system of systems can be found whose aspects cannot be fully captured by any modeling type the statement

is invalidated definitively. While the proof is certainly valid (if it works), it is of little value. Providing one example in which a single model is not enough does not guarantee that other examples will also require multiple models. From the point of engineering design, an experiment that yields more practical results must be found.

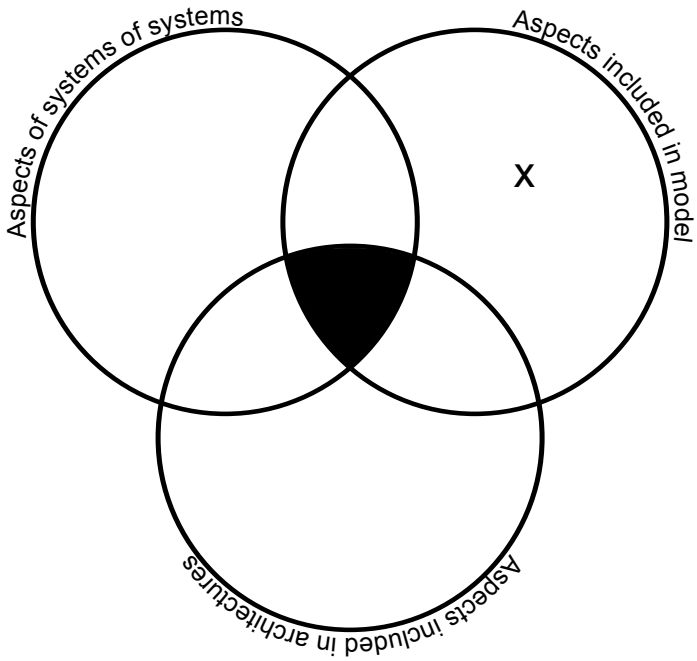


Figure 12: A Venn diagram that represents the idea of architectures capturing all aspects of systems of systems and a model capturing all aspects of the architectures

It also must be said that enumerating all aspects of the systems of systems is not very practical. However, system of systems architecture standards are designed to be flexible for a wide variety of systems of systems. Therefore, if a modeling type can capture all aspects represented in the system of systems architecture, it is able to model the actual system of systems. This reasoning is depicted in Figure 12. White areas depict empty areas, black areas have at least one element in them, and areas denoted with "x" are irrelevant to the problem at hand. One assumption is that system of systems architectures do not capture any aspects beyond what exists for systems of systems.

Venn diagrams are very useful reasoning aids, especially when shading is used to

depict confidence of elements appearing in certain areas. For example, because system of systems architecture standards are designed to be flexible, there is a large certainty that most aspects of systems of systems can be captured by them (Equation 12). Also, because models are created from the architectures as previously discussed, models do not include any aspects beyond what is included in architectures (Equation 13). This is depicted in Figure 13. In this figure, the level of darkness depicts the concentration of different aspects belonging to the systems of systems (white means an empty set).

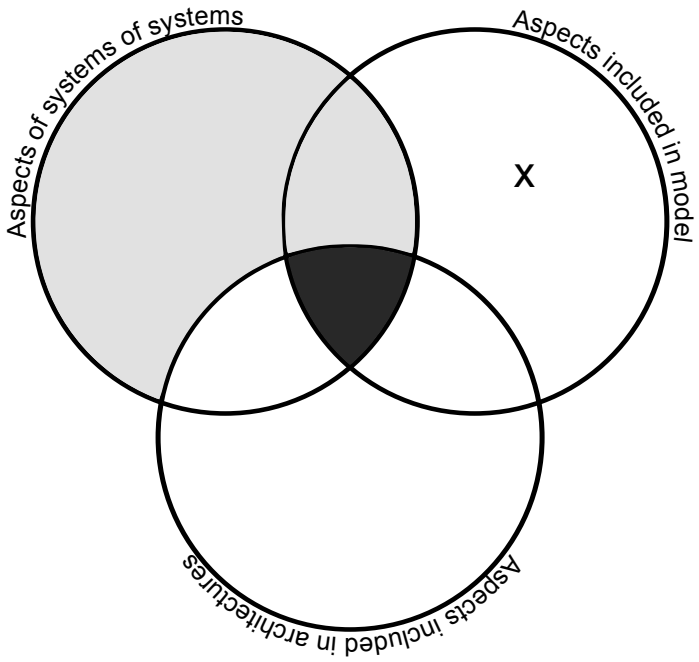


Figure 13: A Venn diagram that represents the idea of architectures capturing most aspects of systems of systems and a model capturing all aspects of the architectures

Unfortunately, Venn diagrams lose their usefulness when more than a few sets are in play. Introducing multiple modeling types, different system of systems, and architecture viewpoints forces equations to be used instead of graphical Venn diagrams. Equations 5–10 introduce the nomenclature for the symbolic development of

a hypothesis on multiple modeling for systems of systems.

$$S = \{\text{systems of systems}\} \tag{5}$$

$$S_i = i\text{th system of systems} \tag{6}$$

$$D = \{\text{DoDAF viewpoints}\} \tag{7}$$

$$D_i = i\text{th DoDAF viewpoint} \tag{8}$$

$$M_i = i\text{th modeling technique} \tag{9}$$

$$A(\cdot) = \{\text{aspects of/captured by } \cdot\} \tag{10}$$

Equations 11–13 show the assumptions made. Equation 11 is a definition of what the function A does. Equation 13 shows a non-essential assumption, but a simplifying one. The "c" in the superscript signifies the complement of the set.

$$A(x_1, \ldots, x_n) = \bigcup_{i=1}^{n} A(x_i) \tag{11}$$

$$A(S)^C \cap A(D) = \emptyset \tag{12}$$

$$A(D)^C \cap A(M) = \emptyset \tag{13}$$

If one single modeling type can capture all aspects of systems of systems, the statements shown in Equations 14–16 must hold. It is important to note that this notation enables the ability to perform examinations with specific modeling types compared to the generic *all modeling types* formulation. This will prove useful at a later stage.

$$A(M_i) \supseteq A(S) \tag{14}$$

$$A(S)^C \cap A(D) = \emptyset \Rightarrow A(S) \supseteq A(D) \tag{15}$$

$$A(M_i) \supseteq A(S) \Rightarrow A(M_i) \supseteq A(D) \tag{16}$$

Equation 14 is a hypothetical statement meaning there is a modeling type that can model all aspects of systems of systems. Equation 15 states that DoDAF viewpoints

do not include any aspects other than the aspects of systems of systems. The third statement follows the first two: if there exists a modeling type that can represent all aspects captured by DoDAF, then there is strong evidence that this model can be used to model all aspects of a system of systems (Equation 14). Therefore, this approach has more value compared to the counterexample, even though the counterexample is more definitive. Also the information gathered for this experiment can be re-used for future system of systems design problems.

If links between aspects of systems of systems, viewpoints, and modeling types can be established, a big step can be taken towards achieving the research objective of methodically selecting modeling types for different system of systems problems.

If one model cannot cover all the aspects described in the system of systems architectures, using multiples can increase the coverage as shown in Equation 17.

$$A(M_1, \ldots, M_n) \supseteq A(M_i) \quad \forall 1 \leq i \leq n \tag{17}$$

This set relationship leads to a hypothesis: *a sufficiently complex system of systems will require more than one modeling technique for analysis.* This hypothesis will be tested indirectly. If some architecture views are used very frequently and these architecture views together require more than one modeling technique to be analyzed, then the argument set forth in the hypothesis will be accepted. Testing this hypothesis requires multiple steps that are formulated next.

3.4.1 Experiment 2

First it will be assumed that only a limited number of architecture views are generated for a system of systems in question which is common practice [6]. The architecture views elected to be used for its description will focus on its aspects that the stakeholders care about the most. Second, modeling types that are able to cover these aspects can be listed as possible solutions to the modeling of the particular system of systems. If a system of systems is adequately represented by a number of DoDAF

viewpoints, then this system of systems can be represented by a model or a set of models that captures the same aspects as the views used to represent the system of systems. This logic is depicted in Equations 18–20.

$$\Gamma = A(D_1, \ldots, D_n) \supseteq A(S_i) \tag{18}$$

$$\Phi = A(M_1, \ldots, M_m) \supseteq A(D_1, \ldots, D_n) \tag{19}$$

$$\Gamma \wedge \Phi \Rightarrow A(M_1, \ldots, M_m) \supseteq A(S_i) \tag{20}$$

If the mapping $A(M_i) \supseteq A(D_1, \ldots, D_n)$ can be established for all i and $D_1, \ldots, D_n$, then subsets of suitable combinations of models can be determined for any system of systems S_j adequately represented by $D_1, \ldots, D_n$. To establish this mapping, each modeling technique will be investigated and the architecture views that can be represented fully by that modeling technique's conceptual model will be identified (i.e., every modeling technique will be tried with every architecture view). Literature has useful examples for these linkages. This is the reverse of the mapping discussed in the earlier induction.

The test can be used on a particular system of systems. However, more information can be gained from a more encompassing study in which not a particular system of systems is studied but a collection of architecture views that are frequently used by system of systems architects. Therefore, a modification is made to the test described earlier: instead of focusing on capturing the aspects of single systems of systems, important aspect of systems of systems in general will be investigated.

The results will be presented in a table with rows as modeling techniques and columns as DoDAF views. One can select several columns representing the more frequently used DoDAF views and construct a sub-matrix with them. If one row in this sub-matrix is fully populated, a single modeling type is enough to model the important aspects of systems of systems. Such a result would invalidate the hypothesis set forth. If the hypothesis is true, the sub-matrix will have no such row.

Additionally, the author expects to find overlapping behavior in architecture views as well as modeling techniques. This behavior will be exploited in two ways. Using the table and a set of system of systems architecture views, alternative sets of modeling techniques for that specific problem can be obtained. For example, a problem may call for an agent-based model or a combined Markov chain and Petri net model to be fully modeled. Also, modeling techniques will exhibit overlap in architecture views, which can then be exploited for calibration and cross-validation efforts. Multiple models can be run and tinkered with until mutually measurable metrics match. This opportunity will be investigated.

3.5 Using the right models for the right problem

The ad hoc practice of system of systems modeling can be observed in examples of similar system of systems problems that have been modeled by different modeling techniques [102, 77, 172]. Although this observation is enough by itself to warrant a deeper study, its reverse is also true: different system of systems problems have been modeled by similar modeling techniques [165, 37]. The author believes that engineers and researchers choose their modeling techniques based on their personal experience and preferences as opposed to what is suitable for the problem at hand. In other words, engineers tend to use modeling techniques they find practical to use (familiarity, availability, ease of use, etc.).

Given the large costs associated with developing systems of systems, considerations of familiarity with the modeling types should not play a role, however. The consequences of choosing an unfit model, extracting the wrong metrics, and basing design decisions on them far outweigh the advantages of practicality in modeling. If engineers choose the modeling technique that match their problem, the analysis through modeling and simulation will be more accurate compared to the engineers selecting models based on their familiarity and availability. Unfortunately, as stated

before, even though system of systems analysis requires diverse modeling techniques, guidance on how to select adequate modeling techniques has been lacking. How can a system of systems engineers decide on a modeling technique if there is no such guidance other than using what has worked for them in the past?

This question aligns exactly with the research goal of developing a method that assists with selecting a modeling type for system of systems analysis. The method uses the information gained from the previous two experiments to determine necessary elements to be represented in the models. Some of these elements cannot be represented with all modeling types, and some modeling types require a large amount of elements to be defined. The modeling approach must balance the requirements of analysis as well as the requirements of the analysis type for a harmonious simulation step for system of systems design.

The method can be inserted into the IPPD flowchart. The engineers start out by creating architectures for the system of systems they are designing. Next, they generate feasible alternatives. In this context, feasible means that they are realizable, not necessarily good. The evaluation step is broken into several sub-steps: selecting model types, creating conceptual models, creating the computer models, and executing the simulations. The results are then fed into a decision-making process in the top-down decision support process. These steps are contrasted with the systems engineering column of the IPPD process in Figure 14.

The next two chapters are included in the document to give an idea of what models and architectures are being considered. Chapter 4 discusses what models are to set up what will be considered a model for system of systems problems. Before coming up with alternative modeling types for system of systems modeling, some guidelines must be set on what a model is. This is so, because there is no accepted modeling type for systems of systems. Chapter 5 will then investigate elements of suitable modeling types and make arguments on how different system of systems elements

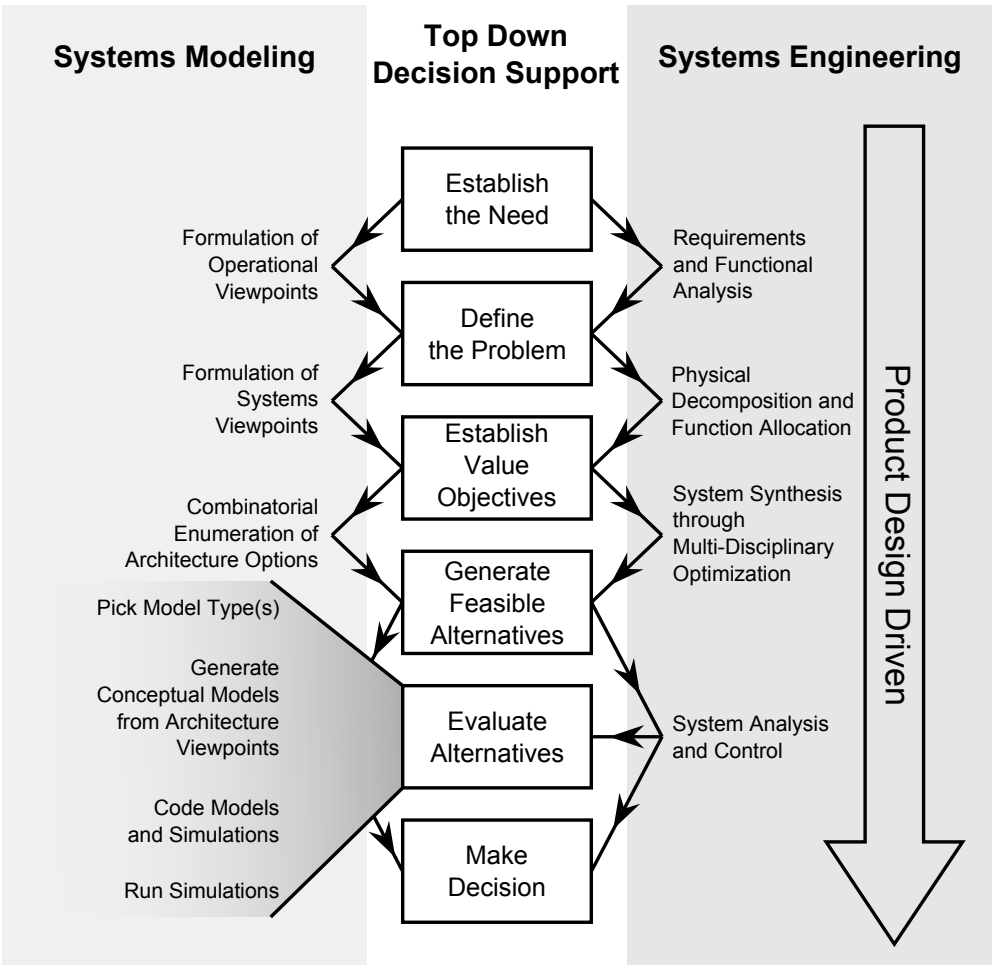


Figure 14: Method of picking a system of systems simulation type

can be modeled by them. Its main purpose is to make analogies based on logical similarities of modeling elements and architectural elements.

CHAPTER IV

ON MODELS

> The purpose of models is not to fit the data but to sharpen the questions.
>
> Samuel Karlin [129]

The research work deals with computer models that are suitable for system of systems analysis. However, before providing background on system of systems modeling techniques, a discussion about modeling in general is required. In engineering fields, the word model is used to describe a large variety of methods, tools, codes, and products. For this reason a philosophical discussion is warranted. A haphazard, *one size fits all* use will certainly create difficulties later; therefore, it is important to establish a good understanding of the concept of a model.

It is important to delineate the sense of the term model used here from the sense the term is used elsewhere in science and engineering. This chapter is broken down into three sections: a discussion on what a model is, a discussion of what a model is not, and finally some guiding principles for further model development for system of systems analysis. The chapter includes answers to the questions listed below.

- What are models?
- What types of models exist?
- Why do scientists and engineers build models?
- How do scientists and engineers build models?
- What are some common misconceptions about models?

4.1 Definitions of a model

A context needs to be established, before going into the definitions, (i.e., the word is analyzed pragmatically). This work treats the word *model* to mean something that helps with understanding, describing, and predicting systems and processes. In this context, *model* is not used to mean

- type of design [176], structural type [176], style of structure or form [176], build [176], make [176], article of apparel of a particular design [176], a motor vehicle of a particular design [176], a vehicle produced in a specified year [176], a specific type or design of clothing or car [87], a style or design of an item [4]
- a person or thing that serves as a pattern or source of inspiration for an artist or writer, one who poses for an artist [87], a person whose profession it is to pose for artists and art-students [176], one that serves as the subject for an artist [4]
- one who is employed to display clothes or to appear in displays of other merchandise [87], a person, ... who is employed to display clothes by wearing them, or to appear in displays of other goods [176], a person employed to display merchandise such as clothing or cosmetics [4]
- a person or thing regarded as worthy of imitation, something perfect of its kind [87], a person or a work, that is proposed or adopted for imitation [176], an examplar [176], one serving as an example to be imitated or compared [4]

Although remotely related to the concept of a model, the meanings listed above are not definitions for a technical model. The following definitions are more aligned with the technical context of a model.

1. "a set of plans for a building to be erected or of drawings to scale for a structure already built" [87], "an architect's set of designs for a projected building" [176]

2. “a representation in three dimensions of some projected or existing structure, or of some material object artificial or natural, showing the proportions and arrangement of its component parts” [176], “an object or figure made in clay, wax, or the like, and intended to be reproduced in a mode durable material” [176], “a small object, usually built to scale, that represents in detail another, often larger object” [4]

3. “description, representation of structure” [176]

4. “an archetypal image or pattern” [176], “archetype” [87], “blueprint” [87], “pattern” [87]

5. “something that accurately resembles something else”’ [176]

6. “a description, a collection of statistical data, or an analogy used to help visualize often in a simplified way something that cannot be directly observed” [87]

7. “a simplified or idealized description or conception of a particular system, situation, or process (often in mathematical terms: so mathematical model) that is put forward as a basis for calculations, predictions, or further investigation” [176], “a schematic description of a system, theory, or phenomenon that accounts for its known or inferred properties and may be used for further study of its characteristics” [4], “a simplified description or conception of a system, used to understand the system or as the basis for further study or investigation of its characteristics” [26], “a pattern, plan, replica, or description designed to show the structure or workings of an object, system or concept” [145]

8. “a system of postulates, data, and inferences presented as a mathematical description of an entity or state of affairs” [183, 138], “a simplified representation of a system or phenomenon, as in the sciences or economics, with any hypotheses

required to describe the system or explain the phenomenon, often mathematically" [61]

9. "such a work or construction used in testing of perfecting a final product" [4]

10. "(in computer programming) a mathematical representation of a process, system, or device" [145], "a computer simulation based on a system" [138]

The definitions above point to several central ideas behind the concept of a model. It can be a mere description of a thing (1–3). In this meaning the model simply describes what that thing is by informing its user about the properties of the thing (e.g., this pen is made out of plastic, its color is blue, its ink is black, and it is 4 inches long). Another meaning of a model could be that it is a blueprint of a thing (1–4). This type of a model can be used to make copies of the thing. A model can also be a simpler representation of a thing (4, 6–9). In this meaning, the model not only describes the thing but also replaces it for examination purposes. A person can examine the model of the thing instead of the real thing because it may be more practical (e.g., a ball-and-stick model of a molecule). Yet another definition would be that a model is a system of attributes, organizing relationships, and functional relationships that describe and explain a thing (3, 7, 10). This meaning puts more information in the model: how the thing works and functions is included in the model. Finally, a model can also be a computer simulation (10), which puts all the above discussed points (i.e., description, representation, and examination) in the virtual domain. A virtual model is a description of a thing in a computer readable format, represented on the screen of the computer, and interacted via a user interface. It is also important to note that computer simulation is widely used as a synonym for computer model.

As previously discussed in Chapter 1, this work assumes that a computer model will be used to analyze the performance of a possible system of systems design alternative. It is, therefore, necessary to establish the concept of a model correctly. If this concept is not understood properly, the models that are built on this dubious foundation will be inferior or simply useless. In the field of engineering many authors use the verification and validation activities to check the correctness of a model. This work acknowledges the steps of verification and validation as critical steps in a model building exercise. However, performing engineering model building based on scientific model building can add to the confidence in the model. Natural sciences have been using models for a very long time with great success; a success that can be enjoyed within engineering as well. The goal here is to learn how to model from the natural sciences. Turner has a summary of several processes for model building for engineering systems in his first two chapters [185].

The above definitions all touch on different aspects of different types of scientific or engineering models. What follows is a discussion on models in science.

4.2 Scientific models

The philosophy of science is the field that deals with concepts, terms, elements, and language that is used in science. Because a model is a scientific concept, philosophy of science is the place to start the discussion on what a scientific model is and why it is constructed. Unsurprisingly, varying definitions of a model are given by many philosophers of science and this section summarizes and organizes them. Carnap's definition of calculus interpretation is used here to seed the discussion.

> "A *calculus* K is constructed and analyzed within syntax in a formal way. As long as we stay in syntax there do not arise questions as to the meaning of the expressions and sentences occurring in K. But, if a calculus K is given, we may go over to semantics and assign designata to signs of K

> and truth-conditions $[S]$ to sentences of K by semantical rules. Hereby sentences of K become interpreted. ... We call S an *interpretation* for K" [45].

An example can help explaining what is meant here. Consider the Newton's Second Law as written in Sentence 7 below.

1. $\underline{F}$ is a vector of size three.
2. The elements of $\underline{F}$ are elements of real numbers.
3. $\underline{p}$ is a vector of size three.
4. The elements of $\underline{p}$ are elements of real numbers.
5. t is a scalar element of an interval of real numbers.
6. $\underline{p}$ is once differentiable by t.
7. $\underline{F}$ is proportional to $\frac{\mathrm{d}\underline{p}}{\mathrm{d}t}$

The sentences above constitute the calculus for Newton's Second Law. However, the law as stated below by Newton is an interpretation of this calculus.

> "The alteration of motion $\left(\frac{\mathrm{d}\underline{p}}{\mathrm{d}t}\right)$ is ever proportional to the motive force impress'd ($\underline{F}$); and is made in the direction of the right line in which that force is impress'd" [151].

Newton's words include meanings, ties to the physical world, and semantics, whereas the sentences initially presented (calculus) do not. The formulas, without the meaning of their terms are literally meaningless, yet still logically deducible or definable. Following this definition, Achinstein offers the following description for a model.

"Let S be a set of statements comprising some theory, and S^* the calculus of this set (what this set becomes by treating each nonlogical predicate constant in S as a predicate variable). Let S' be a set of statements obtained from S^* by substituting an (interpreted) predicate for each predicate variable in S^*. Then S' is a *model* for S^*. Derivatively, we can say that S' is a model for S, and that S' is a model of the items described in S" [13].

These descriptions of models are accurate but not especially useful in building models for engineering work. The interpretation of formulas is a concise definition for models but offers no help in creating models. The above discussion is included in this work for completeness and to serve as a solid foundation for the discussion to come. Perhaps one of the more useful discussions about models in science is given by Heinrich Hertz in his introduction to his book *Die Prinzipien der Mechanik*. What is presented in next is a discussion on what makes models useful and how to define them by their use. The original words in German are supplanted with their English translation by the author.

"Es ist die [...] wichtigste Aufgabe unserer bewussten Naturerkenntnis, dass sie uns befähige, zukünftige Erfahrungen vorauszusehen, um nach dieser Voraussicht unser gegenwärtiges Handeln einrichten zu können. [...] Wir machen uns innere Scheinbilder oder Symbole der äusseren Gegenstände, und zwar machen wir sie von solcher Art, dass die denknotwendigen Folgen der Bilder stets wieder die Bilder seien von den naturnotwendigen Folgen der abgebildeten Gegenstände" [96].

"The most important mission of our scientific knowledge is to predict future experiences in a quantifiable way. Using these predictions, we can decide on which actions to take in the present. We make ourselves images

> or symbols of external objects, and we construct them in a fashion that the necessary consequences of the images are always the images of the scientifically necessary consequences of the external objects we built the images of."

Hertz argues that humans play certain scenarios in their minds to predict the outcomes of actions. The point that Hertz makes is that knowledge about the universe obtained in the past can be used to create a facsimile of the universe in our imagination. Hertz later argues that inside that facsimile, a scientist experiments faster than real-time and can draw conclusions based on these imaginary experiments. In fact, he argues that the main purpose of our knowledge on nature is to be able to make decisions on what to do, before actually performing the action. Holland exemplifies this with a commute scenario in which an alternate route must be found. The driver then *drives* the alternative route in his/her head before actually driving it. According to Holland, a major value of models is that "we can anticipate consequences without becoming involved in time consuming, possibly dangerous, overt actions" [101].

This process is widely known as a Gedankenexperiment or a thought experiment. Hertz argued later in his introduction paragraph that Gedankenexperiments are very similar to models. For this reason they require a special discussion.

Gedankenexperiment is the name of the method to predict future outcomes of actions via imaginary scenarios run in one's head. This method plays a central role in the scientific process. Gedankenexperiments are especially useful during the initial formulating and testing of hypotheses. In fact, a hypothesis that does not pass a Gedankenexperiment test should not be investigated further as a hypothesis must be a claim that a scientist has a reason to believe. Testing hypotheses that are not believable will violate the economy of research. Further down the scientific process, Gedankenexperiments can be used to arrange the experimental setup to test the hypothesis.

One of the major supporters of the Gedankenexperiment was Ernst Mach. In fact, his book has examples of Galileo using proportions to explain the dynamics of falling. The author's favorite example from Galileo's book [76] is the discussion on the ratio of fall velocities of items of different densities. Salviati, who argues for Galilean explanation of dynamics[1], sets up a Gedankenexperiment for Simplicio, who argues for an Aristotelian dynamics[2] view. The experiment is as follows: a wooden ball falls with a velocity of 20 units in air, but in water (here assumed to be 10 times as dense as air) it rises instead of falling. Salviati introduces another ball, considerably heavier than the wooden ball, that falls in water with the velocity of 2 units. Now, this ball must fall with the velocity of 20 units in air; however, it should also fall faster than the wooden ball as it is heavier if Aristotle's two theories are correct. Through this fallacy, Salviati dismantles Simplicio's Aristotelian model of dynamics. This is a perfect example of a Gedankenexperiment to invalidate a theory. The logical analysis of this Gedankenexperiment can be cast in the following way:

H: fall velocity of an object is proportional to its weight

I: objects having different weights do not fall with the same velocity

---

If H is true, then so is I.

But as the evidence shows I is not true.

Therefore, H is not true.

The "*if . . . , then . . .*" structure is used to test hypotheses [95]. It is important to realize that if the evidence is positive, the hypothesis is not necessarily true (fallacy of affirming the consequent). As long as the imaginary scenario leads to a positive evidence, the hypothesis holds. Hypotheses can be falsified rapidly with Gedankenexperiments compared with physical experiments. Because hypothesis testing is aimed

[1]Two objects fall with the same velocity regardless of their weights.

[2]The falling velocity of objects are proportional to their weights. Also, objects fall half as fast in a twice as dense medium.

at trying to falsify hypotheses, Gedankenexperiments are a very useful tool in the creation of valid hypotheses. However, Mach argued that Gedankenexperiments must be later supported by real, physical experiments [128]. This is similar to verification and validation of models and will be discussed in later sections. In hypothesis testing, real experiments must be performed; however, this should not erode the importance of Gedankenexperiments. Real experiments are usually only performed for the one Gedankenexperiment that best tests the hypothesis. Gedankenexperiments can be used to formulate hypotheses, reduce experimentation effort, and guide experimental design. In system architectures, these studies are given as use cases: an imaginary but realistic scenario that is followed until termination and any problems it uncovers must be solved subsequently.

Similar to Gedankenexperiments guiding hypotheses, scientific models guide the development of theories [97]. For example, Bohr's atom theory depicts the hydrogen atom as a massive positively charged nucleus in the middle, and smaller electrons circling it [35]. In his paper, Bohr not only uses theoretical constructs such as charge, mass, frequency, and kinetic energy, but also draws a picture of what an atom would *look* like if one was able to readily observe it. Using the Rutherford theory of an atom, Bohr uses analogies between planetary motion and electron motion, the difference being the different forces responsible for the attraction (gravity in star systems, electrostatic force in atoms). Hesse defines the shared properties between the actual object and the analogous object, *positive analogy*, the differences in the properties, *negative analogy*, and properties unknown whether they are similar or not, *neutral analogies* [97]. She argues that without the neutral analogies, the model is not very predictive or in other words, the incompleteness of a model then should be seen as an exciting topic for further study rather than raising suspicion. Typically, in science, the model focuses on the behavior and configuration, whereas the theory elaborates the reason for the behavior in a mathematical fashion. Models are used to visualize

theories, and even make new predictions [97].

In the light of the discussion above, Achinstein offers a criticism [14] of Hertz's semantical definition of a model presented on page 64. He first divides the claim into five components and describes how these sub-claims do not allow certain models to be classified as models:

1. A model is a set of statements ascribing properties to some object or system.

2. The statements that constitute the model describe some item assumed to be distinct from that of which it is a model.

3. A model is designed to provide an interpretation for an uninterpreted formalism or calculus.

4. A model is always proposed with reference to some theory, the model having the same formal structure (the same calculus) as this theory.

5. A model is an analogy.

The first two arguments are mostly in line with the technical definitions of the word model from dictionaries. However, one difference must be identified: models with the sole purpose of representation do not fit the arguments. Most representational models are not statements. In fact, Achinstein argues that analogies are technically not statements or do not ascribe properties to a system or object, they compare the properties of two distinct systems and note similarities [15]. For further study, the reader is referred to his book. Holland offers a similar perspective: "we discovered mechanisms (gates, pumps, and wheels) and ways of using them to control parts of the world, and we began to model the world with mechanisms" [101]. Models are then parts of the nature represented as mechanisms. Achinstein carefully admits that there is probably not a single unifying set of conditions which can be used to identify a model. However, he proposes the following three types of models [15]:

1. Representational models [147]:

   (a) True: all characteristics of importance (e.g., mass, distance, force) are scaled according to a set scale

   (b) Adequate: only some characteristics of importance are reproduced in the model

   (c) Distorted: characteristics of importance are scaled differently

   (d) Dissimilar: a similarity between two unlike objects is drawn for the characteristics of importance

2. Theoretical models: a set of assumptions about a system or an object

3. Imaginary models: a set of axioms or laws about an imaginary universe, which may or may not be similar to the real universe, but is useful for further study of an object. The modeler does not substantiate the axioms or laws used to construct the model, but substantiates their consequences.

Achinstein puts Murphy's classification of models [147] in his representational models category and adds two other types: theoretical and imaginary. Representational models are used to simply represent the real object or process. They could be scaled replicas that behave similarly (e.g., specimens in mechanical load testing, airfoils in a wind tunnel) or entirely dissimilar things that have similar governing equations (e.g., electromagnetic radiation vs. water waves, radioactive decay vs. death due to horse kicks[3]). Murphy provides a spectrum of similarity between faithfully scaled to dissimilar. He argues that even though engineers desire true models, certain characteristics do not scale linearly in nature and sometimes scaling cannot be accomplished due to the non-existence of materials which have the required characteristics.

[3]Time between events is memoryless, i.e., time between consequent radiation emissions or horse kick deaths does not depend on the previous radiation emissions or deaths due to horse kicks [189]

Dissimilar models can be used to test systems that are very difficult to build but their model is inexpensive. For example, a large vibrating bridge's behavior to excitation can be observed by studying an electrical circuit.

It is important to note that the models Murphy deals with are models of prototypes, and are more relevant in engineering work rather than science. He was interested in representing a system (existing or in design) with a model, hence his focus on scaling and similitude. However, in science models are used more generally. Therefore, Achinstein adds the theoretical and imaginary models to Murphy's representational models [15]. Theoretical models are a set of assumptions in statement form such as "the electron can orbit around the nucleus of a hydrogen atom only in discrete orbits, their allowed angular momentum values are discretized[4]". Such statements are merely assumptions or laws. Usually, no underlying reason is provided. Law and Kelton also draw attention to assumptions being very important for models; however, they argue that such assumptions usually take the form of mathematical and logical relationships [118].

Imaginary models are used to create an imaginary universe that is different than the real one. In this modified universe, certain behaviors can be investigated for consistency. Achinstein provides an example from Maxwell's consideration of whether electromagnetism can be explained by Newton's Laws only. Maxwell does not argue that his explanation is the real explanation, he is trying to figure out whether such an explanation is even possible.

Scientific models are of great importance to the topic of this thesis because they set the underlying principles of how to model and why to model. What follows is a more engineering focused description of models.

[4]This statement is used here as an example of a model that can be obtained from Planck's quantum hypothesis (widely known as the Planck Postulate). Planck argued that the radiation from an atom is always in discretely separated frequencies [161].

4.3 Engineering models

Engineering models are usually within Achinstein's representational models category [15, 147]. The goal of modeling in engineering is to replace the real system with a prototype, scaled model, or replica for probing, testing, analysis, and experimentation. Therefore, engineering models share many properties with the scientific models. One of the most important shared properties is the similarity between the modeled system and the model of the characteristics that are important for analysis purposes.

Consider a wind tunnel model of an airliner as an example. The model is constructed to replicate the mechanics of flight to a degree of accuracy. The aerospace test engineer can confidently predict the real system's flight performance from the data gathered from the model. This scale model does not share all of the properties with the real airplane (e.g., weight, density, material). However, certain characteristics are carefully scaled (e.g., external geometry, airflow velocity) to match similarity parameters such as Mach and Reynolds numbers that would be experienced by the actual design during operation. There are some parameters that are not scaled (e.g., angle of attack and sideslip angle). If scaling is done properly, the measurements of the aerodynamic force and moment coefficients experienced by the model will accurately match the coefficients that will be experienced by the actual airplane in operation.

Using the model, the aerospace engineer can predict the design's performance confidently to make design decisions (e.g., change design parameters, adjust expected performance, confirm design goals). In aerospace engineering—as well as some other engineering fields—wrong design decisions can be incredibly costly. The design engineers must guide the design of the system towards a more capable, but simultaneously cheaper final solution. This can be achieved by accurately predicting the impact of design decisions on the effectiveness of the final design (i.e., trade studies) [135]. For example, the advantages and disadvantages of increasing the wing's aspect ratio ($\mathcal{R}$)

must be known to the engineers before they can make decisions on that parameter.

Because most modeling in engineering is related to design, an interesting difference from scientific models must be discussed. In science, models are based on observations, they are hypotheses in the form of "if model is true, then the observations are consistent with the model's consequences". However, in engineering design, models are used in the absence of the real systems, objects, and processes. The engineers do not have the final product in place. Otherwise, the design activity would be pointless[5]. During the design phase, the model must be constructed in a way that predicts the performance of the yet-non-existent system it is representing to a degree of accuracy. This is different than scientific models in the following ways discussed next.

4.3.1 Non-existence of the real system

The represented system cannot be observed, not because the observation is not physically possible but simply because the system does not exist. This issue forces the engineers to model using many assumptions and characterizations. Because the design is not fully defined, the engineers do not know many design parameters of the final design, and as such they cannot model the system fully. Another consequence of the *real* system's non-existence is that there are a large number of decisions the design engineers must make to define the large number of *free* design parameters. These parameters are fixed via trade offs. Practically, this is not possible in the beginning of the design phase, when there are too many parameters to be dealt with. Therefore, elimination or consolidation of design parameters is a required step in the early phases of design. Lastly, the model cannot be truly validated as experimentation with the real system is not yet possible [119].

Returning to the airliner example from before, the location of windows, control surfaces, and internal hydraulics piping are usually not known at the beginning stages

[5]This is only true in the context of design. If the real system exists and the engineer is only trying to gain a better understanding of it via modeling, the engineering activity is not design.

of the design. Therefore, the effects of many components must either be approximated or completely ignored via simplifying assumptions based on past experience (e.g., the majority of the lift will be generated by the wing). The engineers then proceed to model at a reduced level, i.e., a level where they can observe the components of the system. In the airliner design example, the engineers can model the aircraft lift as the lift generated by the airflow over the wing. The inaccuracies caused by neglecting subsystems can be reduced by a calibration factor at a later design phase. This type of simplification allows the engineers to rapidly prototype scaled replicas wing design alternatives without getting bogged down on the tiny details.

4.3.2 Replacement of the real system

The models used in engineering design are not explanations of how the system works, but rather an approximate replica for the system during experimentation. Most engineering models make use of scientific knowledge about well-understood phenomena. The model is made not to understand the system or process better, but for improving it. This aspect of an engineering model is entirely different from scientific models. Scientific models are used to develop new theories, engineering models are used to develop new systems and processes.

For example, Minsky offers a definition of a model that underlines the replacement of the system: "To an observer B, an object A* is a model of an object A to the extent that B can use A* to answer questions that interest him about A" [140]. Minksy's definition is less about the explanatory function of a model. He stresses that there is an underlying understanding of the object A in his argument; however, his definition is lacking this point.

It must be pointed out that many engineering models are developed in a scientific way. For example, Prandtl found a component of drag, arising solely due to lift using Kutta-Joukowski's theorem. Prandtl showed that the pressure difference between the

upper and lower surface of the wing will create a circular flow from bottom to top. He modeled this as a vortex in the three dimensional vector field (representing velocity). From this vortex, he calculated the changes in the sectional circulation at different locations on the wing and finally derived Equation 21 for induced drag coefficient [6].

$$c_{Di} = \frac{c_L^2 S}{\pi b^2} = \frac{c_L^2}{\pi \mathcal{AR}} \tag{21}$$

This equation was developed just like any other scientific model: starting with a previous theory (Kutta-Joukowski), changing an element in it (finite wing instead of infinite), representing the important elements by other constructs (in this case velocity as a vector field), predicting what the consequences of the model are (induced drag being proportional to lift squared), and finally testing it with real experiments to validate the model. Prandtl compared his theoretical induced drag results with experimental results for a rectangular wing of aspect ratio 5. The comparison can be seen in Figure 15 [184]. The disagreements are due to viscous and separation effects, which are not accounted for by induced drag.

In modeling, it is important to explain a certain phenomenon and resist the urge for fitting to the experimental data. As can be seen in Figure 15, Prandtl's induced drag does not match experimental data very closely. Having observed the discrepancy, Prandtl explains how his induced drag is only a part of the total drag. This begs the question: "how can one really measure only the induced drag"? One cannot simply subtract zero-lift drag from the total drag to find induced drag, because the increase of drag due to changing angle of attack is not necessarily due to drag due to lift. One could use the same wing with endplates and subtract the drag of that from the drag found without endplates (after removing the drag of the endplates themselves), but still the separation profiles will be different. It must be accepted that some models cannot be directly validated by real experiments. This supports the argument made

[6] Equation 21 uses a translated English notation. Prandtl originally published it with German subscripts [163].

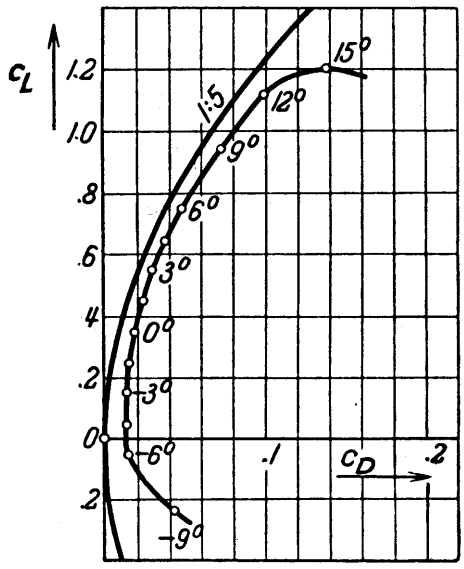


Figure 15: Prandtl's comparison between induced drag and experimental data [184]

earlier that models can be used as hypotheses, which cannot be directly validated.

Prandtl was able to derive his induced drag mathematically from circulation. If one accepts the generation of lift is due to circulation, induced drag can be demonstrated via mathematical proof. However, circulation is a model for the generation of lift and is not necessarily correct. The main underlying assumption with these theories is that an inviscid flow with no pressure change due to altitude can be represented as a conservative vector field. That analogy is the actual model, and the equations that follow are the Potential Flow Theory. Per Hesse's argument discussed earlier, models are important in developing theories [97]. Achinstein might go as far as calling this a theoretical model [15]; however, Prandtl's work does not stop at making assumptions and considering consequences. Discussions such as this make it clear that defining and classifying models is indeed a difficult endeavor. In the field of system architectures and system simulations, it is no surprise that almost every product is called a "model".

4.3.3 Providing accurate observations

The main point of an engineering model is to provide accurate observations in absence of the real system. This is somewhat different than the scientific models. Scientific models strive not only for accuracy but also understanding and possibly generalization. However, engineering models have a smaller scope of replicating behavior of an engineered system. Nevertheless, such observations can be treated as predictions because the real system does not exist yet. Therefore, using Hertz' argument on page 66, the purpose of engineering models is to predict system performance in a quantifiable way to make informed design decisions before building the systems themselves. Engineering models can be used to cut down on prototyping efforts [118].

Replicating every experiment performed by models with a real-life experiments is usually pointless in engineering design. For example, if a bridge is being built, not every analysis simulation must be repeated by building many bridges and trying them out. However, several carefully selected experiments should be performed in real-life to gain confidence in the results of the model. This procedure is widely known as *model validation* in the literature [23, 46, 119]. The knowledge of model validity is important in engineering design, because the main point of engineering modeling is providing accurate observations without experimenting with the real system.

Minsky argues that a model of a thing is good, if the model answers the modeler's questions about that thing, and if the answer of the model matches the answer the modeler would get if the modeler experimented with the actual thing itself [140]. The importance of the agreement between the real world and the model is similar to Hertz' discussion on page 66. This author is of the opinion that Hertz defined the agreement in a more precise way, but Minksy's introduction of *questions to be answered* is important. A model is not meant to replace the system in scrutiny entirely but only partially, which should be enough to make observations related to the study at hand. For this reason, using adequate, distorted, and dissimilar models

as defined by Murphy on page 70 is acceptable.

Murphy offers a definition for a model as "a device which is so related to a physical system that observations on the model may be used to predict accurately the performance of the physical system in the desired respect" [147]. This author believes that Murphy's definition goes too far into the scaled physical model domain to be applicable in many other engineering fields. Can the analogy between an inviscid flow and a conservative vector field be called a *device*? A better wording for the definition of a model is required; however, the idea of "accurate prediction in the desired respect" is noteworthy.

In science, models are created as part of the learning process. Observations are classified and hypotheses are generated for possible explanations. However, in engineering (especially in design) models are generated solely to provide accurate performance predictions for systems that do not exist yet. Since the creation of the model is necessitated by making correct design decisions, a required accuracy can be stated ahead of model development such as "the model will predict aircraft lift within 5% error". Such tolerances can be used to determine whether a model is good enough to be used in design. Following this discussion on models, a working definition and examples of computer models will be discussed as shown in Figure 16.

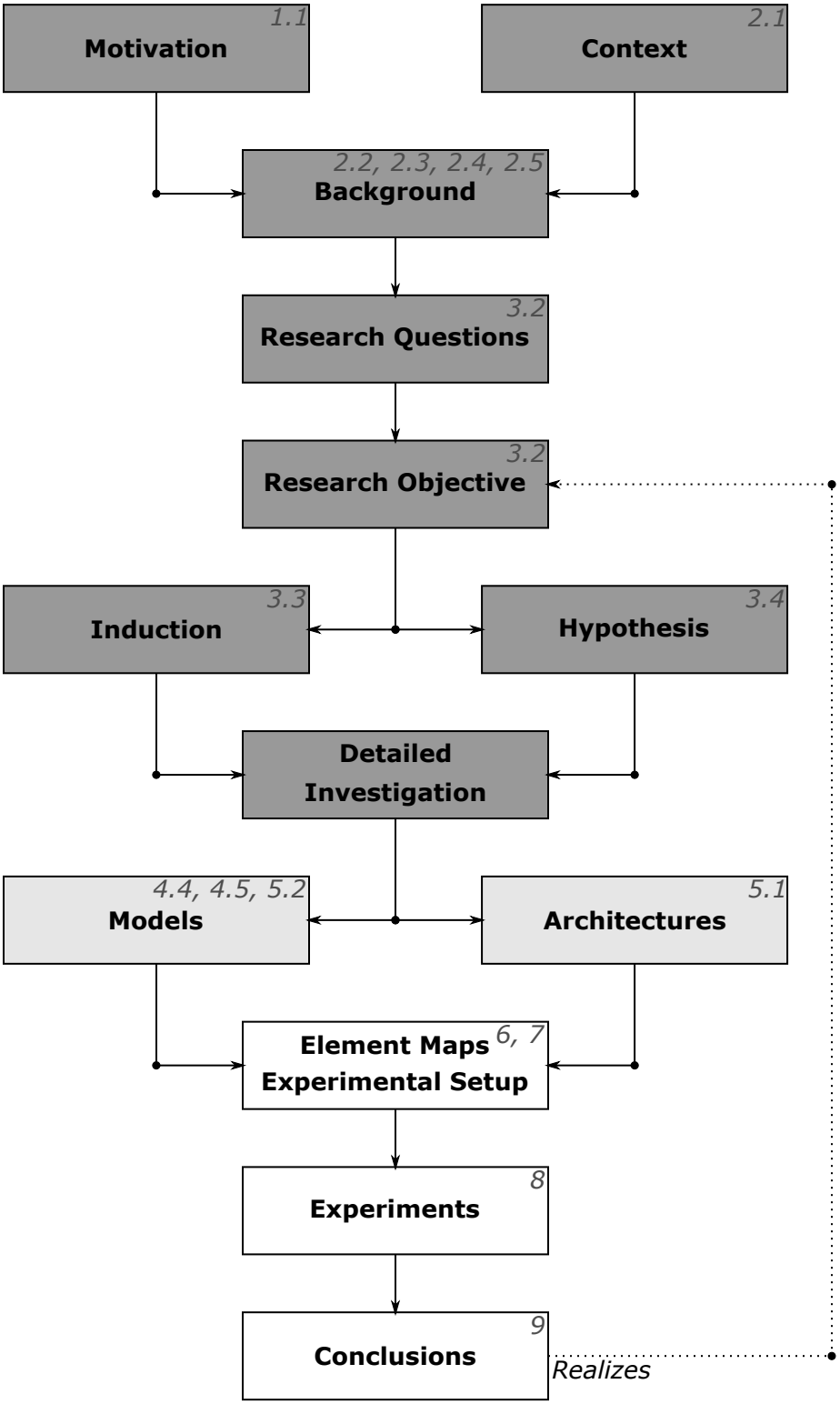


Figure 16: Working definition of a model and types of computer models to be investigated next.

4.4 Computer models

Computer models are one step away from mental models. Whereas most models discussed so far are used in Gedankenexperiments to guide theory development or design systems, models that are executed on computers are used on computers to do exactly the same. The immediate advantages are obvious:

- computers can keep track of a large number of entities,
- computers can provide quantitative results,
- computers are very efficient in performing repetitive operations (e.g., iterations, repetitions, recursions),
- computer experiments can be automated,
- computational power is cheap, and is becoming cheaper still.

Computers can not only hold a large number of data in a structured fashion, but also put that data in fast random access memory modules for fast execution rates. This enables computer to work with a very large information sets, track many *objects* in a model, and compute the interactions between the objects. As the number of objects grow, human minds struggle with the interactions between such objects. This is mainly due to the number of interactions growing much faster than the number of objects as given in Equation 22, where n is the number of objects and I_n is the number of interactions between n objects. This growth is depicted in Figure 17 as well. Computers are also very efficient in performing repetitive tasks, and the large numbers of objects and interactions can be easily processed by computers.

$$I_n = \frac{n(n-1)}{2} \tag{22}$$

Because computers can be automated to execute many experiments, these experiments can be embedded within optimization problems or database generation.

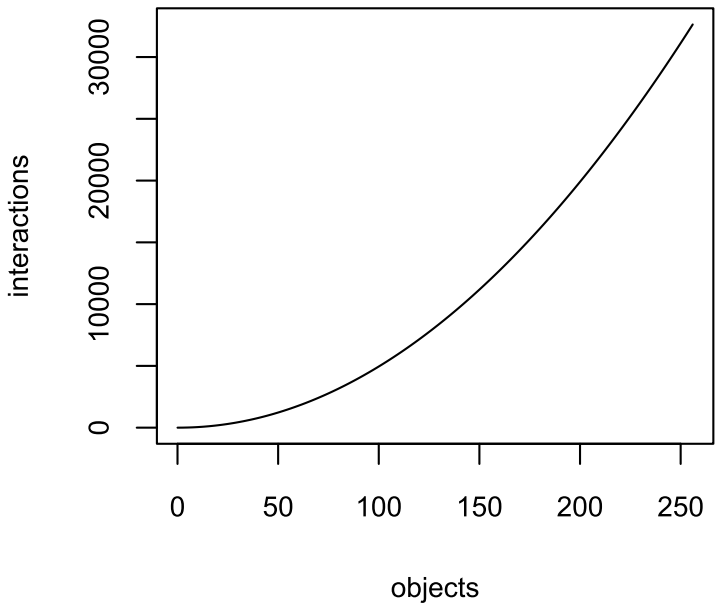


Figure 17: Number of interactions between objects grow very fast

Entire design problems can be performed by computers iteratively with little human intervention. In fact, one reason that nations are racing with each other to build the fastest supercomputer is to tackle larger problems such as economic policy making [74], galaxy formation [112], molecule folding [122], weather patterns [167], and nuclear reaction simulations [39]. Given the trend of transistors becoming cheaper, more densely arranged, and less power hungry [173], computer modeling has a very promising future.

The promises come with some drawbacks however. Computer models share an extra step during their formation. As discussed before, models are possible explanations of how systems work. These explanations are formed inside human brains, and need to be translated into a machine readable language. Apart from translations not being perfect, there is also always a chance of making mistakes. Computer models must be checked thoroughly to find such coding mistakes as well as checking for agreement between simulation outputs and the real world. This process is what is known as

verification and validation of computer models. The reason why it has two names is that verification is inherently separate from validation.

In order to explain the difference between verification and validation, conceptual models must be introduced. Conceptual models are like models and also the equations related to them. Just as theories and models cannot be easily separated into exclusive halves, conceptual model models and their machine interpreters are difficult to separate. This inseparability has several different incarnations. The most important one is their conceptual model cannot be developed without a machine interpreter in mind.

Also, because machine interpreters, that are rather generic, are not developed with or for models, interpretor limitations apply directly to models that are meant to be simulated by them. These two issues are discussed next.

Figure 18 shows the process of modeling for a finite wing with an aspect ratio equal to 5, an $\mathcal{R}$ value that is not too small. First, the modelers envision a flow around a section of this wing (airfoil) as a conservative vector field, à la Prandtl. Later, they expand upon this view by replacing the entire wing by some vector filaments. Now that the wing is abstracted into mathematics, using nothing but equations, they can derive the drag due to lift for a finite wing. Then, they correctly enter their equation using the proper syntax required for a computation engine (R, in this instance). Finally, by running their code, they can observe some trends, which can be compared to their real data as shown in Figure 15. The modelers must check their code (verification), mathematical derivation (verification), and their output's accuracy (validation).

Another modeler can now come in and use Prandtl's mathematical model and develop equations of motion for an aircraft using Newton's Second Law, which is depicted in Figure 19. Here the forces due to aerodynamic effects are modeled as vectors at quarter chord location and the mass of the wing is reduced to a point

mass. From Newton's Second Law, the forces can be used to derive the kinetics of the system. Later, this model must be translated into machine language. Because the modeler chose MATLAB to simulate this system, MATLAB's syntax must be used. MATLAB's *ode45* solver deals with first-order ordinary differential equations [133]; therefore, Newton's Law (a second order differential equation) must first be reduced to a first-order equation. This is a classic example of how conceptual models require some modification before being translated into machine code. Once the code is prepared, the simulation can be run and the results observed.

If this model was being developed in a more graphical simulation software, the development of Newton's Second Law and its translation into a larger system of first order differential equations would not have been necessary. In such a simulation, the definitions of attachments (boundary conditions), rigid body properties (mass and inertia), and forces (application point, direction, and strength) would be sufficient. If the model was used in conjunction with another model that solves for the aerodynamic forces and moments, even less of a definition would have been needed.

These cases illustrate the influence of the selected simulation software (or in general, simulation engine) on the development of the model. However, nature is clearly insensitive to such choices. Ideally, the creation and the solution of the equations is kept separate. In the case of computer simulations, such a separation is unfortunately impractical. At each step of computer modeling, the modeler must adhere to the rules of the simulation engine to make an executable model. Such decisions detract from the actual modeling goal, and may result in mismatches with the reality. Therefore, verification and validation exercises are more critical for computer modeling efforts.

4.4.1 Verification and validation of computer models

Balci argues that accuracy, execution efficiency, maintainability, portability, reusability, and usability (easy-to-use user interface) are good qualities for a modeling and

simulation effort [23]. Depending on the modeling and simulation effort each one of these qualities can capture the limelight; however, in all modeling and simulation efforts accuracy must play a major role. Without accuracy, modeling is not only useless but also potentially misleading. In computer modeling, the accuracy of models is checked and provided by testing, verification, validation, and accreditation activities [23, 119, 168]. Unless a computer model is verified and validated, its results cannot be depended upon as there is no reason for trusting the model [168]. This part discusses verification and validation of computer simulations and how they compare to accuracy checking for scientific models.

Verification is determining that a simulation computer program performs as intended [119]. The *intent* here is not to match reality with the model, but to match the logic in the modeler's mind with the computer model. The modeler's understanding of what is happening in reality is usually referred to as a *conceptual model* [119, 169]. A computer model can be perfectly verified, yet still lack accuracy when compared to the real system. For example, Prandtl's Lifting Line Theorem is perfectly verified via mathematical proof, but it still lacks the major effect of viscosity on drag; therefore the model is not accurate. A similar situation may occur in computer models. The code may be absolutely correct, but the conceptual understanding of the system may have been wrong. The resulting computer model and its execution would also be wrong (not accurate).

Verification then deals with the correctness of the coding activity. Balci refers to this activity as a translation from conceptual ideas to computer code [23]. In his words, "model verification deals with building the model *right*" [23]. These words are likely to be misinterpreted. If one builds a model right, why would the model be wrong? This author believes that Balci meant that model verification deals with coding the computer implementation of the model as intended. The meaning is not altered by this modification, only sharpened. In fact, Balci argues that verification

measures only the accuracy of converting a model representation from a flowchart or a similar medium into an executable computer program.

The meaning of verification has a similar meaning in software engineering. According to the IEEE standards, in the software engineering context, verification is the process of attaining proof of correctness [1]. For example, a database record retriever takes in a key, searches for that key in a database, then returns the data attached to that key within the database. A formal proof of correctness makes sure that the returned data is truly attached to the key for all possible key entries. It can also deal with cases such as what to return if the key is not found in the database. However, it does not check whether the developer's ideas are translated into the *program*. As long as the program does its job correctly for every situation bounded by the requirements imposed on it before the development began, the program is correct. An alternative definition offered by Hetzel and IEEE is evaluation performed at the end of a phase with the objective of ensuring that the requirements established during the previous phase have been met [1, 98]. Perhaps a more fitting definition is given by Davis, "verification is the process of determining that a model implementation accurately represents the developer's conceptual description and specifications" [57].

Computer model verification has a scientific model counterpart. Prandtl's Lifting Line Theorem models inviscid flow as a conservative vector field. But as Hertz said, the necessary consequences of the models must match the necessary consequences of the modeled objects. Therefore, Prandtl had to develop his model mathematically to reach its consequences. These consequences (such as induced drag) then are compared to the consequences of flows over finite wings. Checking over the mathematical development of Prandtl's theorem can be compared to the verification of computer models. The inability to compare models with reality directly necessitates verification. This means that every model must be developed in a logical, abstracted, mathematical way in order to be compared with reality.

The comparison of the model's logical consequences with the consequences of the nature is called *validation*. This is the most important for a model to be acceptable, and it is in a similar form as hypothesis testing. In the context of computer models, there are various definitions in the literature for validation. Carson states that validation is "the process of ensuring sufficient accuracy for a model" [46]. With a definition such as this, it may seem that verification is a part of validation or that validation is enough and verification is not necessary. However, this is not true. Law and Kelton argue that validation is concerned only with determining whether the conceptual model is an accurate representation of the system under study [119]. In practice, however, the separation of conceptual model validation and output matching is not attainable per Hertz' discussion presented on page 66.

Take a statistical regression that has good predictive power as an example. It can be shown readily that it is accurate. However, is it a good model? The author believes that a statistical regression is a sophisticated calibration of a very flexible function and not a scientific model. Calibrations, although useful in certain scenarios, do not guarantee a representative model [168]. In fact, statistical regressions are not analogies, do not have any explanatory value, and are geared to match results directly. They do not have an image with a necessary consequence that matches with the consequences of reality, they directly match with the consequences of reality. Therefore, in a scientific sense, regressions are not models and validation by itself is not sufficient in determining accuracy of a model.

Every modeling and simulation effort must have a clear goal from the beginning [23, 120, 169]. Predetermined modeling goals protect the modeling effort from becoming modeling for modeling's sake. A set of requirements—a software specification—are required for a computer model to satisfy [98], and a modeling effort can be ended, once all the requirements are satisfied. Since good requirements are unambiguous and verifiable [93], a level of accuracy deemed adequate for the simulation study at hand

should be set before modeling starts. During and after the modeling process, each requirement can be tested. If all tests are passed, validation is practically complete. In this view, validation has a binary solution: "the model is valid", or "the model is not valid" [168]. A fitting definition for validation in this case is offered by Balci, who defines validation as substantiating that the model, within its domain of applicability, behaves with satisfactory accuracy consistent with the modeling and simulation objectives [23].

Validation in the scientific sense, however, cannot be finished [119, 168]. There are many scientific reasons for this. First, a thing can only behave exactly like another thing, if and only if it is that thing. A model can only accurately replace a system in a limited operational scenario. Second, there are no proofs in science, only theories that are thoroughly tested. And third, if a model is exactly like a system, then experimenting with the model is as difficult as experimenting with the real system, which renders the modeling activity superfluous. Therefore, validation must be regarded as a confidence building exercise [22]. It can only be considered complete for the purpose of a study, but not for the actual model. This fact has profound effects on multi-purpose models, re-use of models, and certification of models.

As enough confidence is gained for a model, it can be certified by responsible entities for specific uses. This step is referred to as *certification* or *accreditation*. DOD defines accreditation as the official certification that a model or simulation is acceptable for use for a specific purpose [3]. IEEE defines software certification as a written guarantee that a system or component complies with its specified requirements and is acceptable for operational use [1]. Accreditation or certification is required for models to be used officially.

Validation or certification is not always required for a model to be useful. Oftentimes model development occurs without the presence of real quantitative data. For example, Galilei has not measured the gravitational accelerations of objects to

create a model for constant accelerated falling. Similarly, Prandtl did not measure the induced drag effects before working on the Lifting Line Theory. An industrial engineer does not need to know exact work completion time statistics to model a work station. Numerical data can be entered into the model afterwards for validation purposes. Without this data, a verified model can still generate useful general information for decision making (e.g., higher $\mathcal{R}$ results in smaller c_{Di}); however, it cannot be used in a context that requires high accuracy. This also means that validation not only checks the accuracy of the consequences of the model, but also the accuracy of the model's inputs. Input validation is usually dealt with separately before model validation.

In summary, the observations leading to model development are usually very qualitative (e.g., objects seem to reach the ground at the same instant when released from the same height). The models are created using analogies to drive generalization. Model development can proceed without real accurate input data, but cannot finish without it. The goal of modeling is to make new observations (predictions) by experimenting with the model, and without accurate predictions, the model would be of limited use. With the insights gained from this discussion a working definition of a model for the rest of the work is now given.

4.5 Working definition of a model

After the discussion of what a model is and how they are constructed, a working definition of a model for the purposes of this work is defined here. The definition is specific to models that are used to

- do science, that is to test hypotheses and create more observations,
- perform analyses on a computer (sometimes in an automated fashion),
- experiment with non-existent systems,

- design new engineering systems, and
- help making engineering decisions.

The definition draws from Hertz' view of what a model is used for and how it is checked with Murphy's and Minsky's additions of dissimilarity and scoped accuracy. The definition is given as necessary conditions for a thing to be called a model.

1. A model is based on an analogy (e.g., inviscid flow is just like a conservative vector field).
2. A model is used to predict the performance of a system (e.g., $^{L}/_{D}$).
3. The necessary consequences of a model are closely matched with the consequences of the real system (e.g., increasing $\mathcal{AR}$ decreases c_{Di}).
4. A model is not a replica of the real system; it is dissimilar in irrelevant properties (e.g., the mathematical construct of a conservative flow field does not contain molecules)
5. A model is a good model if it can predict relevant performance values the engineer is interested in; other performance values do not have to match reality (e.g., yawing motion is irrelevant in a three-degrees-of-freedom study).
6. A model of a system must explain the internal mechanics of that system, not just match its consequences (e.g., simple statistical regressions are not models).
7. A model must be executable in order to allow experimentation and making of new observations (e.g., mere descriptions of systems are not models)

When considering the dictionary definition of a model, descriptions and statistical regressions actually do fit within the concept of a model. In fact, statistical models can replace systems for experimentation purposes with some limitations. However,

statistical models are not thought or conceptualized by someone and they do not explain anything. They are mere trends—a clever summary of data so to speak—and there is little value to be gained from them to lead into the generation of more models. Regressions have hypothesis testing embedded within them; however, these hypotheses are not of the form Hertz discussed. They do not involve testing consequences, only characterizations.

Using this definition of a model, a large variety of modeling paradigms are introduced in the next chapter and their applicability on system of systems problems are discussed. The models introduced are presented in an order based on a taxonomy of type. Each type is validated by the definition of the model presented above.

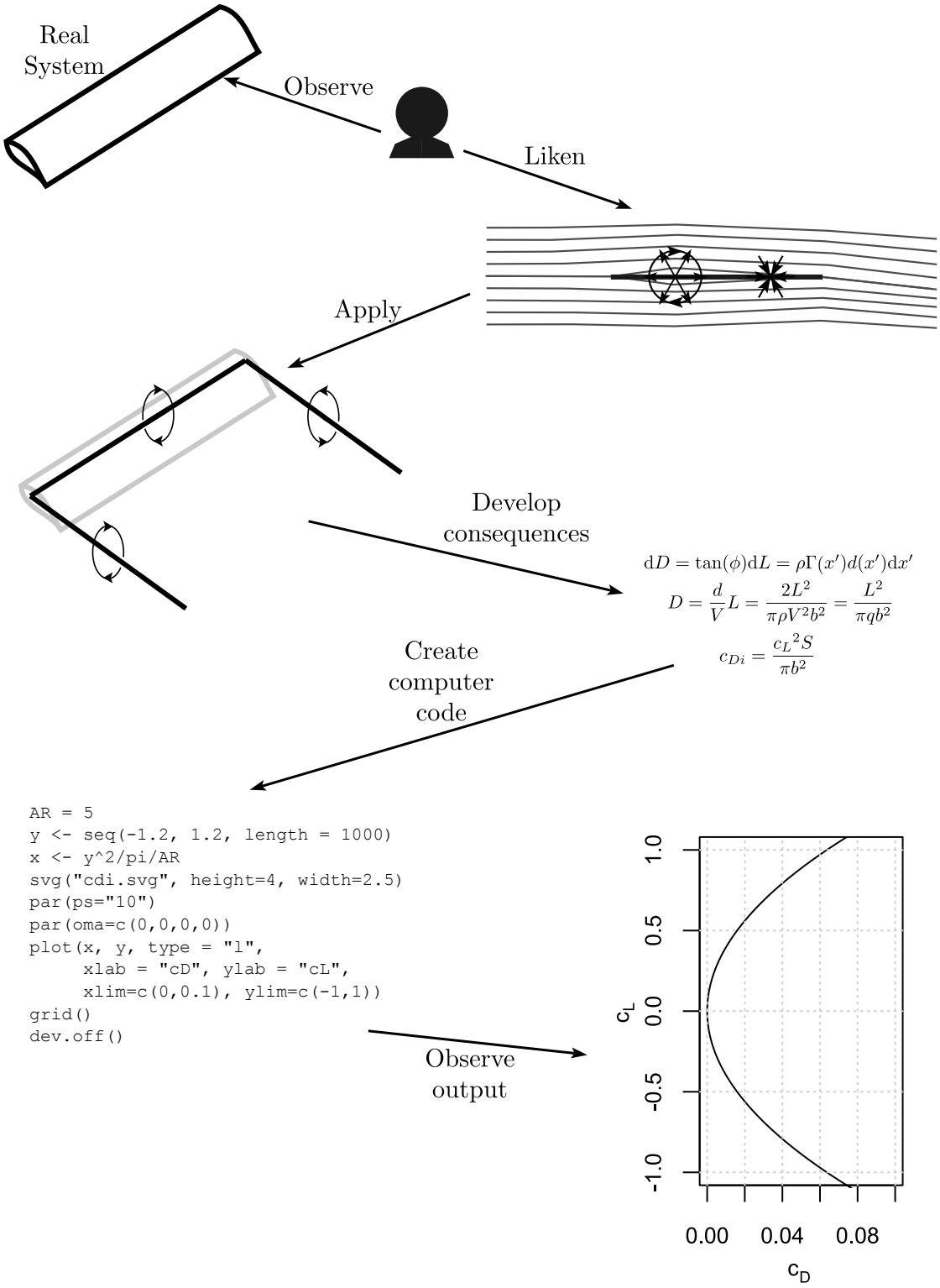


Figure 18: An example of a computer model evolution

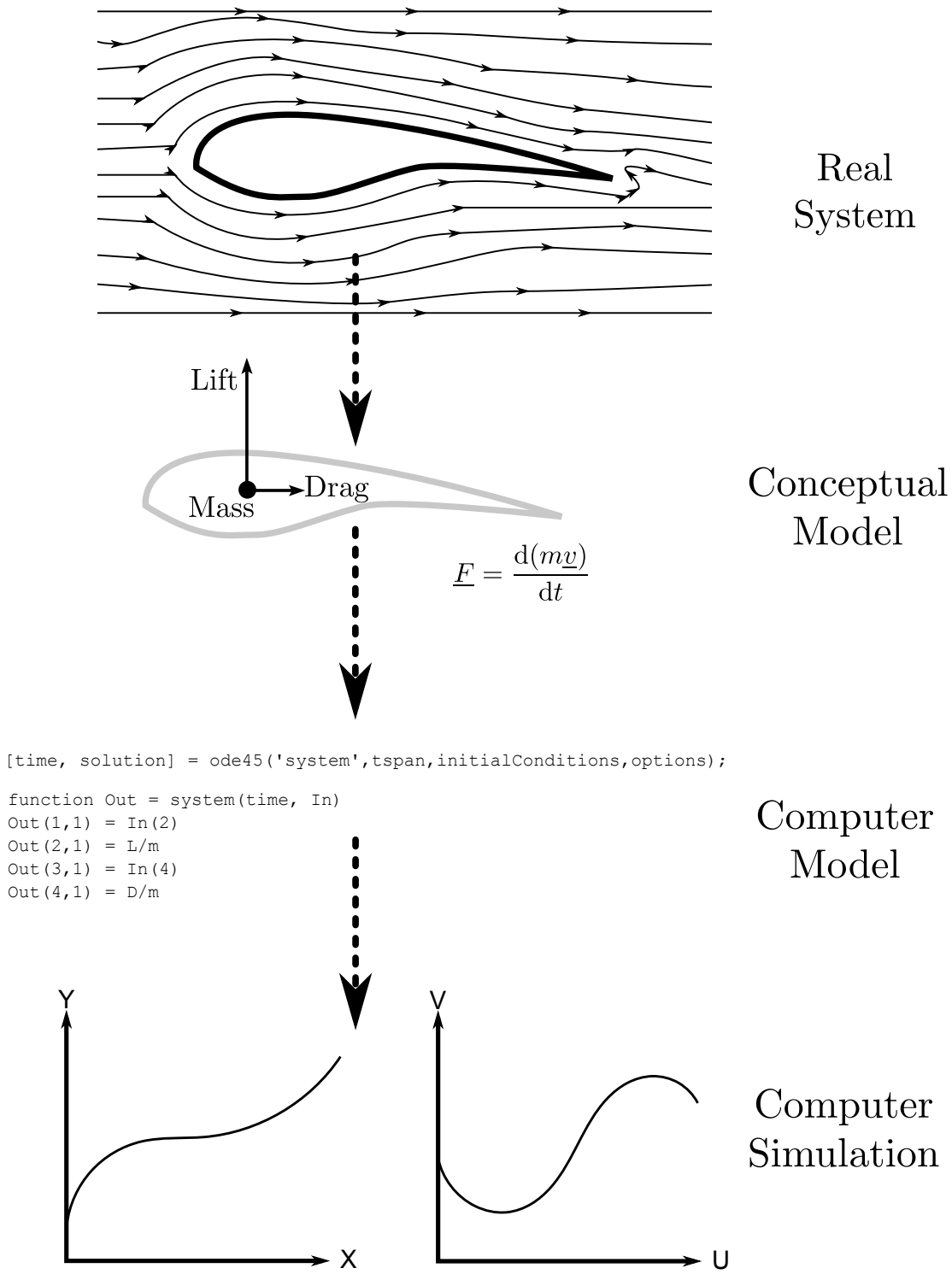


Figure 19: An example of a computer model evolution

CHAPTER V

SYSTEM OF SYSTEMS MODELING

> Ich ekelte mich oft vor den Menschen, die fließend ihre Muttersprache sprachen. Sie machten den Eindruck, daß sie nichts anderes denken und spüren konnten als das, was ihre Sprache ihnen so schnell und bereitwillig anbietet[1].
>
> Yōko Tawada [180]

System of systems engineering deals with mostly large-scale engineering. At the very least, the name of *system of systems* suggests the engineering of multiple systems. In most cases, system of systems thinking is applied when complicated systems cannot perform the necessary functions. If one can create a single system[2] that can perform all the necessary functions to create an effect or enable a capability, there may be little reason to even do system systems engineering. Admittedly, there are exceptions to this rule, but usually complicated systems are more efficient than their complex system of systems counterparts just because they can be engineered easier. This is due to the fact that the interactions between system components are easier to deal with than interactions between multiple systems with their own decision making abilities.

[1] Author's translation: I loathed people who spoke their native language fluently. They made the impression, that they could not think or feel anything other than what their language offered easily and quickly.

[2] Here a single system is not used to mean a single platform. Not all dispersed but connected systems deserve the name system of systems. For example, the global positioning satellites are hardly a system of systems. The components of the system do not act on their own, and as single satellites they are not useful themselves. The reader is reminded that the system of systems definition used in this work required self-capable and useful systems to be integrated into a larger group of systems to deliver unique capabilities. Communication, therefore, is not collaboration, and a single system can be geographically dispersed.

Two important aspects of system of systems were identified in the preceding paragraph: large scale and looser interactions between components. If the interactions between components were very well defined, systems of systems would have been called *systems*. These two aspects require the use of different kinds of modeling compared to models used for physics-based interactions between system components and between the system and the environment. Physics-based models are usually accompanied by mathematical formulations in the form of differential equations. However, such a mathematical order is found rarely in system of systems formulations. The interactions seen in system of systems are more condition-based; therefore, algorithms are better suited to represent them.

The utility of mathematical models are not lost, however. If the modelers can use mathematics to model a system of systems, they should definitely explore that path further. Mathematical models are very powerful and tend to have good scalability characteristics. The sections of this chapter will introduce many modeling techniques that are useful for system of systems modeling efforts. Some of these techniques are based purely on mathematical formulations. However, the limitations of such models will be made clear as well, such as having to make numerous assumptions about the system.

It is important to realize that simplifying assumptions are required for all types of modeling. In Chapter 4, it was mentioned that Achinstein actually argued that the assumptions are the model itself [15]. From that argument, it follows that making no assumptions means making no models, which demonstrates the importance of *reasonable* assumptions. The modeler must be very careful not to make too many assumptions and must always substantiate why a certain assumptions was made. So, it is clear that incorrect assumptions lead to incorrect models (incorrect conceptual models, to be specific). Still, an *incorrect* model can supply much needed information on the design. As George Box argued: "all models are wrong; the practical question

is how wrong do they have to be to not be useful" [38]. If a decision maker is uneasy making decisions based on incomplete information, design process will not benefit from a modeling and simulation step. In fact, the only option in such a scenario is prototyping many alternative solutions and picking the best performer based on real operational tests. In system of systems problems, such approaches are impractical.

5.1 Design description

Before reaching the modeling step of system of systems engineering, a common way of representing a design is needed. Here, a *model* in the descriptive sense is needed. The discussion in Chapter 4 led to the conclusion that such descriptions should not be called models for the rest of the work. Therefore, they will be called *design descriptions* and they simply describe the system as it is without making any statements on why the system works the way it works. For example, "the fire extinguisher is red" is a design description, "the mail is brought to the sorting facility via trucks" is also a design description. These descriptions do not include the information why red is the best choice for a fire extinguisher or why it takes 3–5 business days to deliver mail. They are simply statements that can be used to model the system but are not themselves models. Such descriptions are not only important in the modeling phase, but also for documentation and communication purposes as well.

Many design descriptions exist. Any computer-aided design product is a design description encoded in some electronic format. Such products can have some extra information that may qualify them to be categorized as models but in most cases they are simple descriptions of the systems they represent on the computer. Blueprints, system architectures, circuit diagrams, floor plans, furniture assembly manuals are all design descriptions.

Such documents are used by various entities such as engineers, designers, technicians, users, certifiers, etc. Because of this, they are usually in a very human accessible

format: print. Print documents are by definition static, meaning that they do not show the changes to a system's states. Some of these architecture views include enough information to show the change, but the views themselves do not show it. In the systems engineering field these documents are known as system architectures. They are the main document types used for communicating and describing designs to various stakeholders. Each system architecture includes several views, viewpoints, models, diagrams, tables, lists, and dictionaries. There is no common convention of creating such architectures but there are some frameworks on what to include such as DoDAF [59] (other military frameworks include Ministry of Defence Architecture Framework [105], NATO Architecture Framework [10]), Automotive Architecture Framework [41], and Zachman Framework [203]. Apart from the frameworks, a language is also required to create the system architectures. Using an architecture description language, a system architecture can be saved in a consistent format. Not all languages and frameworks are compatible, and usually, a framework is developed with a language specific to serve its framework. Examples of system architecture description languages include UML [5], Systems Modeling Language [8], and Integration Definition for Function Modeling [2].

Until recently, system architectures have mainly been used for documentation purposes. Static hard or soft copy documents have been used as data containers for system architectures. Such containers include technical reports, presentation slides, manuals, doctrine documents, graphs and charts, etc. However, the documentation view is changing. If such documents were made in a computer readable fashion, models created to analyze designs can use them as inputs in support for the simulation. This is the main idea behind the DOD's push for executable architectures. DoDAF is a good framework for developing system architectures; however, it is only slowly evolving to include features that help with analysis steps [143]. In this context, an executable architecture provides the means to conduct dynamic analysis of a system,

and is emerging as a supporting methodology [202].

The shift from documentation to dynamic simulation is a symbiotic development between the system architect and the system modeler and a significant driver for developments such as Foundational Subset for Executable UML (fUML) by the Object Management Group (OMG) [182]. In a system design context, the system architect's goal is to describe the system at the level of detail where past decisions are included. On the other hand, the system modeler's goal is to generate knowledge and aid future decision making. These two processes mimic the iterative design process of analysis and decisions as discussed in Chapter 1 where AoA were discussed.

Similar to proper documentation, dynamic simulation-ready system architectures require a detailed and complete description of the systems it is representing. Such an architecture would contain enough information to create unambiguous inputs to a simulation model. However, it must be remembered that in this thesis the goal of modeling and simulation is to make design decisions. A significant amount of information is missing from the system description during this phase. System or system of systems architectures at this phase have many degrees of freedom, and they cannot be completed fully. To be useful in design, system architectures must be flexible in the amount of information required to describe a design.

The two requirements discussed above are somewhat contradictory. One requires a detailed and complete description, the other requires a workable architecture even if information is missing. Actually, this domain is where modeling and simulation shines. Enough description is necessary to create models, but no more. Since models are simplifications of the reality, they require less information than other efforts such as acquiring, building, and operating. Because this thesis deals with the modeling and simulation of systems of systems and entities interested in acquiring and operating them are mandating the use of system architectures during development, it will be assumed that an architectural description on the design will be available for or being

worked on in parallel to the modeling effort.

It is interesting to compare the definitions of models and architectures. The first point of Achinstein's five-point definition [14] for a model states that a model is a set of statements ascribing properties to some system. Architectural descriptions of systems are not models, but are close because they describe properties instead of ascribing them. Also, artifacts of system architectures simplify the system by only looking at it from a specific perspective and eliminating many other details about the system in that view. For example, neglecting other properties of the constituent systems, a system of systems can be shown as a connectivity network. Architectures descriptions are similar to models, but for the purposes of this work, are not.

5.2 Models for system of systems

Given a system of systems design description, the next step in design is to find its performance. The analysis step is needed to make further design decisions. For example, two designs can be analyzed independently and the better one can be picked for further development. The computer models are relevant during this phase of the system design.

The analysis step can be summarized as finding a function that links design description/variables to the design's abilities/performance. Given design parameters, the analysis must return design performance. The design phase has an analysis phase in it, because without an analysis of design alternatives, decisions cannot be based on quantitative metrics. Unfortunately, there are no closed-form analytic formulas that perform system integration and report on the capability of a system of systems. Because no such equation exists, the system of systems engineer must use computer models to map design variables (inputs) to design performances (outputs).

Figure 20 repeats the generic design decision making process based on the IPPD

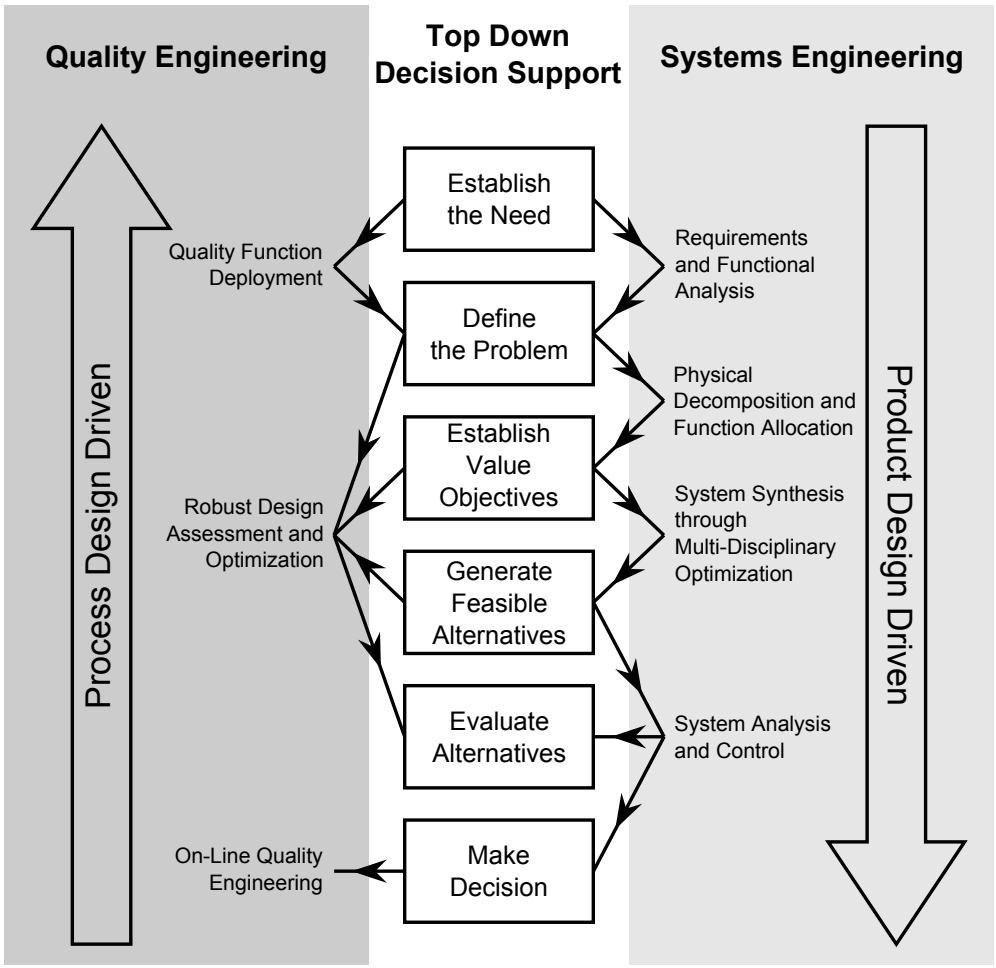


Figure 20: Georgia Tech Integrated Product and Process Development Methodology [174] (Reproduced)

methodology [174] first introduced in Chapter 2. Architecting the system starts during the requirements and functional analysis. At this stage, the architecture only includes functional decomposition (in DoDAF terms mostly the OVs). In the physical decomposition and functional allocation step, systems and physical considerations are added. This step serves as combinatorially generating possible architectures, whereas system synthesis happens through an optimization. The optimization has two goals: eliminating unfeasible designs and finding good compromises between performance and cost. In order to reach both goals, the optimization step requires several functions from each discipline to be optimized, and that function is the analysis step in design. Finally, a more holistic product-level analysis is performed during evaluation

of the alternatives. That is yet another analysis step, in which a calculation of some kind must be performed to find the best possible alternative. In conclusion, modeling can be applied at almost all stages of design because design is tightly coupled with analysis.

Because executable architectures are also recommended by the DoDAF, concurrent development of architectures and computer models are desirable. Following the IPPD methodology, as soon as the problem is defined, architecting and modeling activities can start. Here, architectures and models are separated from each other. Some authors call architectures—or parts of thereof—models, but in this work they will not be called models because they are not analogies based on the discussion in Chapter 4. However, it must be stated that many architecture frameworks were created with certain modeling formalisms in mind, and the separating line between description and modeling is usually blurred. Generating executable models from the architectures automatically using this blur will be detailed in the later chapters.

DoDAF's executable architectures are system architectures supplanted with models which can be executed on a computer and provide more than descriptive information on the design of the system of systems. In order for this to work, architectural elements must be likened to modeling elements. This will be discussed in the following sections. These modeling elements must then be organized in a machine readable format for simulation engines to execute them. Computer executed models (i.e., simulations) are the only practical means of calculating system of systems performance in a consistent, automated, and rapid manner. The results of the simulation are analyzed and put back into the architecture, thereby providing more than descriptive information about the system of systems it represents.

Such model execution does not have to be on-line. If the simulation execution takes a considerable amount of time, the designer may have to wait for the performance values to appear. There are ways to alleviate this problem. First, a simpler more

analytic model can be used if possible. If that is not possible, statistical regressions based on data generated by more complicated slow-executing simulations can be created and included in the architecture. Because statistical regressions are accurate within limited regions based on the data used to create them, the designer must be careful not to extrapolate from that region [148].

There are a limited number of quantitative modeling paradigms fitting to the system of systems problem. The modeling paradigms used to analyze system of systems found in the literature are listed below.

- Graph theory [64, 91, 24, 88, 50, 75, 115, 77]
- Probability theory [175, 77]
- System dynamics [29, 44, 43, 88]
- Markov chains [50, 88, 17, 131]
- Petri nets [75, 88, 172, 109, 79, 124]
- Queueing theory [17, 65]
- Discrete event [88, 102, 20, 69, 37]
- Agent-based [24, 77, 172, 99, 204]

The discussion of each modeling paradigm is organized in a taxonomy of models based on their execution, type of metrics used, and inclusion of probabilistic effects as shown in Figure 21. The reader will realize that some modeling types can be under multiple sections of the taxonomy[3] and when this is the case, only the differences from the previous case will be discussed. For example, there are two types of Markov chains: continuous-time and discrete step. Therefore, Markov chains appear under at

[3]the taxonomy is a valid taxonomy, but modeling paradigms are flexible and can be configured to be under one or the other; a single model is never under multiple categories

least two categories in the taxonomy; however, Markov chain models under different categories differ from each other and are simulated entirely differently.

Because this work deals with quantitative models, qualitative models will not be investigated. Qualitative models are very important in engineering, especially during the early stages of design. In fact, all quantitative models start as qualitative characterizations (e.g., objects tend to fall, heavier objects fall roughly with the same velocity as lighter objects, objects fall slower in water). The transition from experience-based or subject matter expert qualitative models to quantitative models is an important topic that is being tackled by other authors [90, 67, 66, 136]. However, in this work, the focus will be on quantitative models.

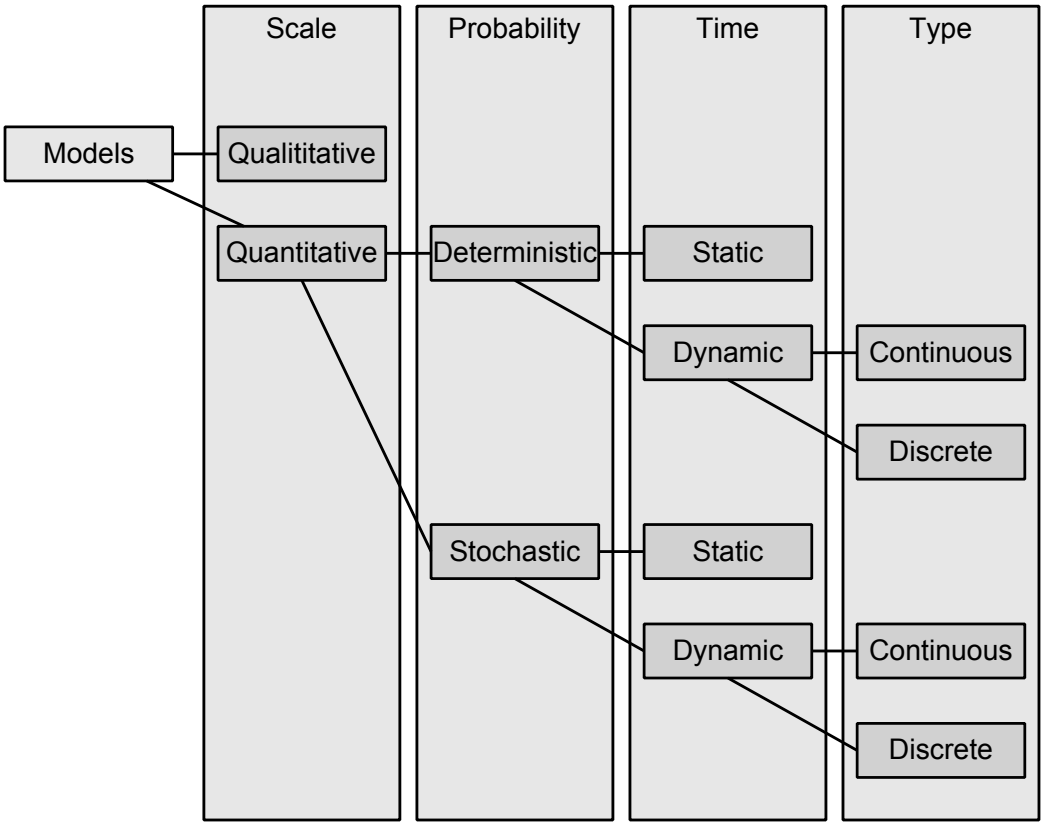


Figure 21: Modeling Taxonomy adapted from Burbank et al.[42]

5.3 Deterministic and static models

5.3.1 Graph theory

In the simplest sense a mathematical graph is a collection of vertices and edges. These vertices and edges have some meaning assigned to them. This way mathematical graphs can represent chains of events (function/process decomposition), locations and distances (pathfinding), or systems and connections (networks), etc. See Figures 23, 24, 25, and 26 for four distinct examples of scheduling, pathfinding, hierarchy, and networking with graphs. The flexibility in the graph theory approach makes an analogy between the multitude of systems in a system of system and networks. A network is a basic organizing structure that is easily understood. Graph theory uses the network abstraction to represent almost anything with separate elements and the connections between those elements.

Graph: A graph is an ordered pair $G = (V, E)$ comprising a set V of vertices (nodes) and a set E of edges (arcs) which are two-element subsets of V.

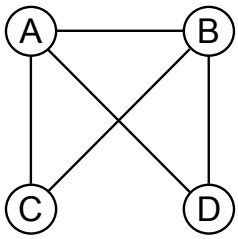


Figure 22: A mathematical graph

An example graph is shown in Figure 22. Its corresponding mathematical notation is given in Equations 23–25.

$$G = (V, E) \tag{23}$$

$$V = \{A, B, C, D\} \tag{24}$$

$$E = \{(A, B), (A, C), (B, D), (B, C), (A, D)\} \tag{25}$$

Besides its flexibility, this abstraction can be analyzed using analytic methods. Graph theory is a field in mathematics and for many of its measures, a closed-form analytic calculation is possible. As discussed earlier, analytic calculations are desirable in models due to their ease in verification and computation. Most graph theory calculations use linear algebra; however, some analyses still require an algorithmic approach.

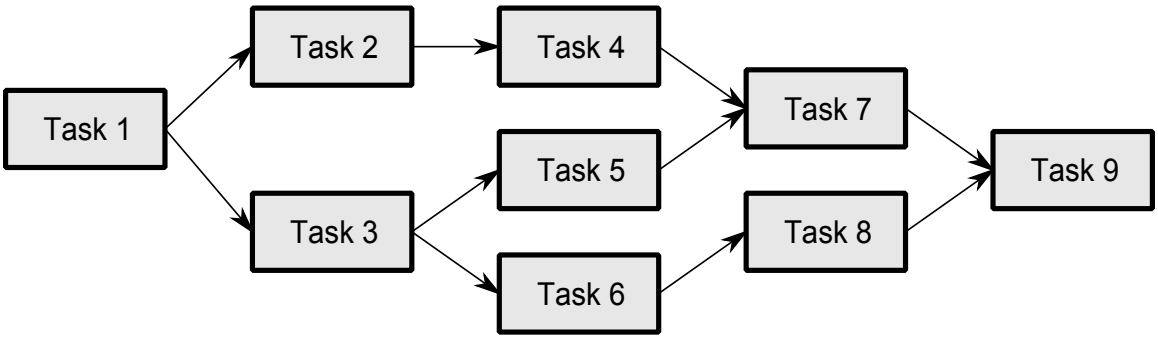


Figure 23: Scheduling with graphs

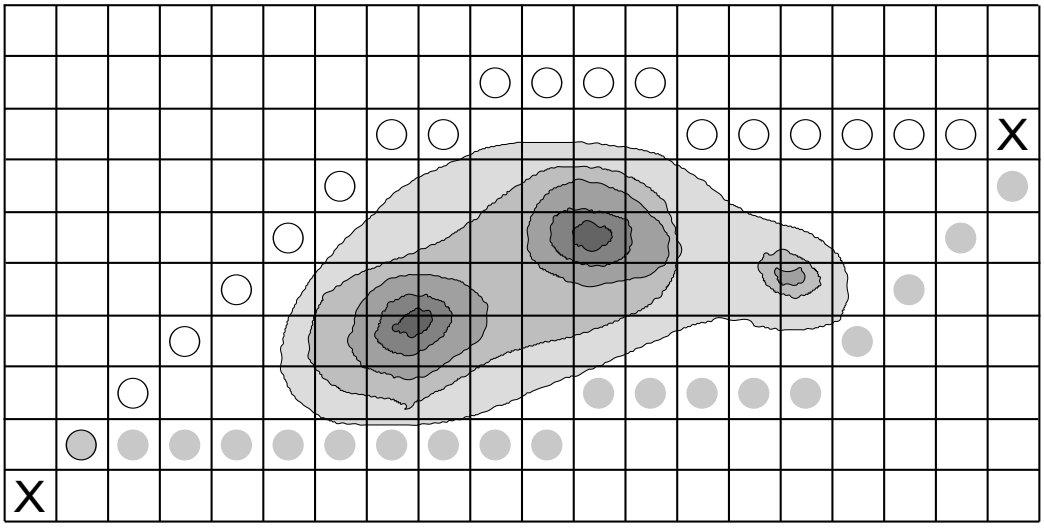


Figure 24: Pathfinding with graphs

The rules of graph edges are as follows. The edges are always placed between two vertices. They are never terminated before reaching a another vertex. An edge in a mathematical graph can be either bidirectional or unidirectional. Bidirectional edges can be decomposed into two directional edges. An edge can also have have a value,

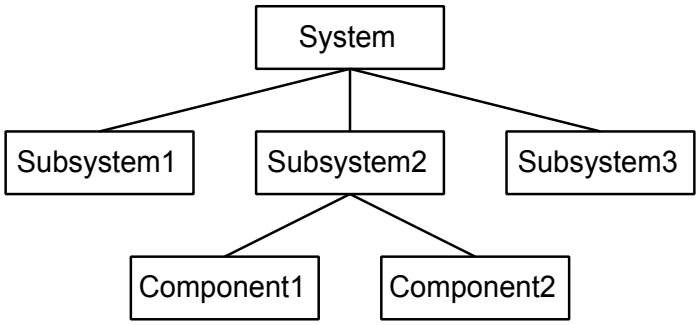


Figure 25: Trees with graphs

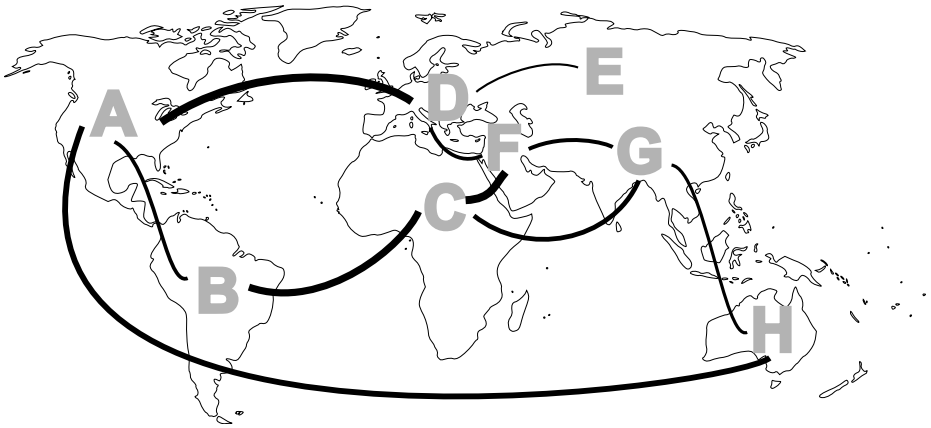


Figure 26: Attributed connectivity with graphs (Map source: Wikimedia)

or a list of values assigned to it. Vertices are not assigned any values to them; they can only be labeled.

In terms of analogies, graph vertices can represent systems and edges can represent connectivities between them. Because the edges can be directed, they can represent one-way or two-way radio communications. The weight used on the edge, can represent many different connectivity metrics between the two vertices it touches (e.g., bandwidth, travel time, delay, availability rules). This analogy is commonly used in system of systems analysis [24] to measure centrality, geodesic distance, vulnerability, maximum data flow, and similar metrics of the network. There are algorithms that deal with each of such metrics, such as max-flow min-cut for throughput, A* for shortest path, among many others.

Alternatively, vertices can represent different tasks that a system of systems must

perform, similar to what is shown in Figure 23. If all tasks are not required, the graph can be used to find the shortest path to the final state. If all tasks are required, the graph can be used to analyze centrality of subtasks as well as the cyclicity and complexity of the larger task network [24, 64].

5.3.2 Probabilistic calculations

In mathematical graphs, the edges and vertices are used to construct networks of connected elements. A similar approach is used in probability calculations; however, the assignment of meaning to the modeling elements is different. Probability calculations can be represented with mathematical graphs, but in a probability network each vertex represents a probability of a certain task, process, action, etc. successfully occurring. In the terminology of probability theory these tasks, processes, and actions are called *events*. The probability of an entire network of events happening can be determined by stringing these probabilities together as a chain of events.

This theme will repeat itself in the later modeling discussions. Because mathematical graphs are very flexible, many different ideas can be represented by them. Probability calculations, Markov chains, Petri nets, and discrete event simulations can be represented in structures that are very similar to graphs. However, the analogies in each modeling paradigm work differently. This makes them different models and their analysis is vastly different.

In such probability calculations, the connections only provide information about the structure/layout of events. The numerical data included in the model is held entirely within the nodes. This data is supplied in the form of conditional probabilities (i.e., the probability of the event happening given that all previous events have happened). The aggregate probability is then calculated using Fermat's Principle of Conjunctive Probability. The principle is stated below. An example in graphical form is given in Figure 27 and corresponding equation is shown in Equation 26.

Fermat's Principle of Conjunctive Probability: the probability that two separate events will both arrive is hk, where h is the probability that the first event will arrive, and k is the probability that the second will arrive when the first event is known to have arrived [196].

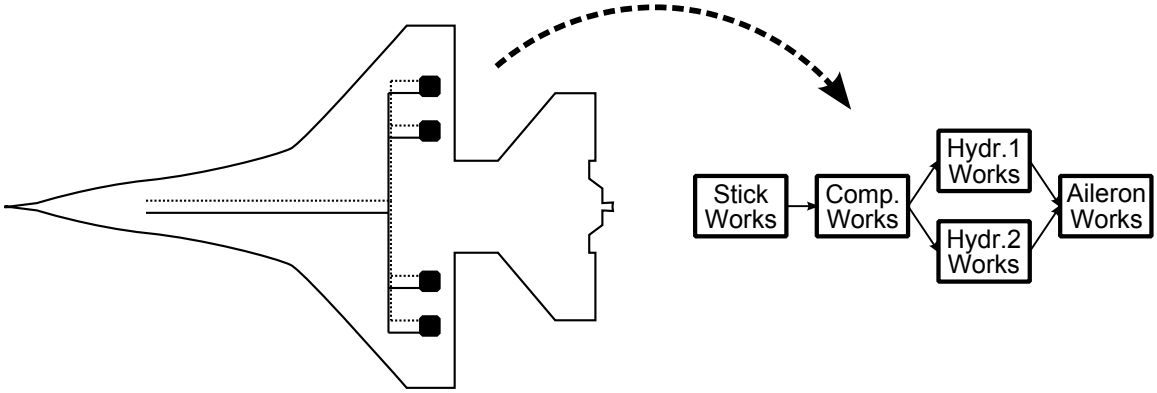


Figure 27: Conditional probability example: aircraft control surface actuation system

$$\begin{aligned} P(\text{Aircraft Rolls}) = & P(\text{Stick})P(\text{Computer}|\text{Stick}) \\ & (1-(1-P(\text{Hyd}_1|\text{Stick}\cap\text{Computer})) \\ & (1-P(\text{Hyd}_2|\text{Stick}\cap\text{Computer}))) \\ & P(\text{Actuator}|\text{Stick}\cap\text{Computer}\cap(\text{Hyd}_1\cup\text{Hyd}_2)) \end{aligned} \quad (26)$$

Single probability calculations of this kind are simple and computationally easy to perform. The difficulty in aggregating the conditional probabilities lies in the complexity of the network. The network needs to be converted into an equation which is a difficult task if the size of the network exceeds a handful of nodes. The example given in Figure 27 has only 5 nodes yet Equation 26 has grown to 8 terms due to the redundant nature of hydraulics system represented in the model. In a more complex network, equations will grow even faster, which renders writing the equation for the network in its entirety difficult.

Given the large scale nature of system of systems engineering problems, it is desirable to have scalable modeling techniques. In order to solve the complexity and

scalability problem, a recursive algorithm is proposed here. Recursive algorithms call on themselves with a smaller problem, and the first call is finished once the smaller problem reaches a base case. A structure is recursive if the shape of the whole recurs in the shape of the parts and this property allows recursive algorithms to turn highly complex software products into smaller and much more easily understood functions [81].

In order for recursion to work, the probability network must be reducible to simpler and fundamental elements, and then re-aggregated for the probabilities to be evaluated recursively. Step by step aggregation of probabilities is given in Figure 28.

Finally, when the irreducible elements are reached, calculation rules governing these elemental forms must be found. In probability calculations, there are only two of such elemental forms: *series* and *parallel*. Layouts and combinations of series and parallel forms are shown in Figure 29.

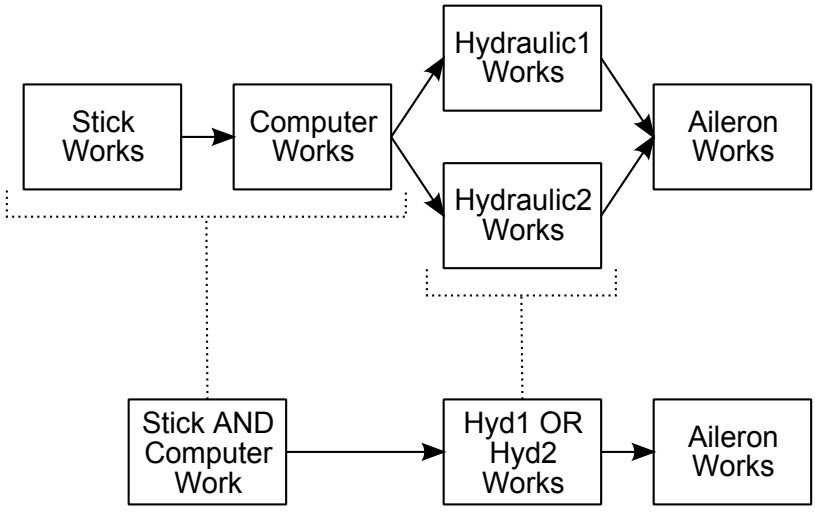


Figure 28: Combination of probabilities in the control surface actuation example

Calculating the aggregate probability of success of events in series form is done by multiplying the conditional probabilities, which can be derived in a few steps from set theory fundamentals. In order for the events in a series network to execute successfully, all events must execute successfully. Equation 27 shows the probability

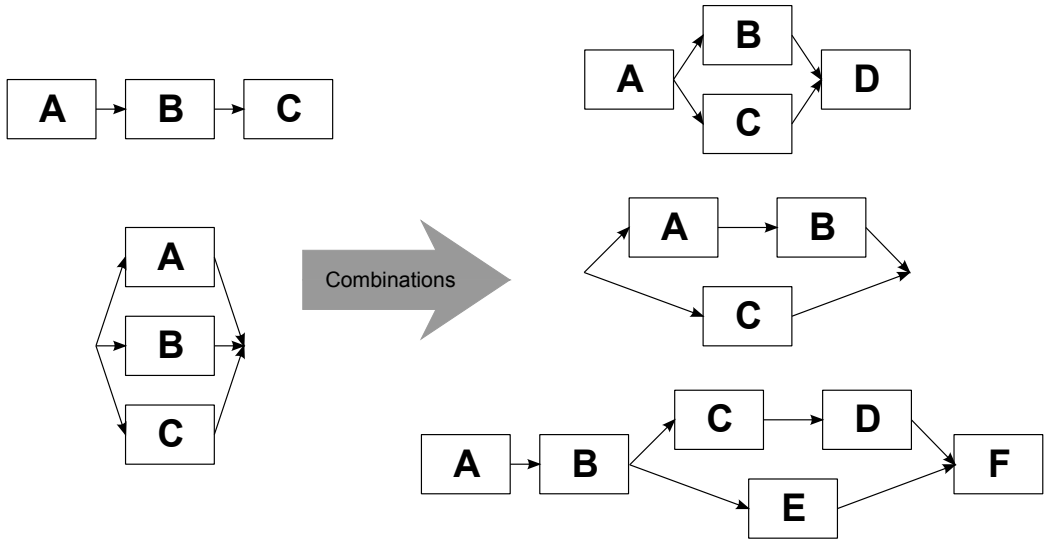


Figure 29: Series and parallel layout and possible combinations thereof

of events A and B happening. A series with n events has the probability shown in Equation 28. If the events are independent from each other, the equation can be simplified to Equation 30 using the definition of independence, which is given in Equation 29.

$$P(A \cap B) = P(A)P(B|A) \tag{27}$$

$$P(E_1 \cap E_2 \cap \ldots \cap E_i \cap \ldots \cap E_n) = P(E_1) \prod_{i=2}^{n} P(E_i|E_1 \cap E_2 \cap \ldots \cap E_{i-1}) \tag{28}$$

$$A \perp\!\!\!\perp B \iff P(A|B) = P(A) \tag{29}$$

$$P(E_1 \cap E_2 \cap \ldots \cap E_i \cap \ldots \cap E_n) = \prod_{i=1}^{n} P(E_i) \iff A \perp\!\!\!\perp B \tag{30}$$

Calculating the aggregate probability of success of events in parallel form is done by reformulating the question to "the probability of failure of all events". This reformulation uses the fact that all events in the parallel network structure must fail for the network event to fail as a whole, which is of course a probability calculation in series formulated in an equation. Equation 31 shows the idea of using an intersection instead of a union for the set of events. Equation 32 generalizes the idea to multiple events in a set. Finally, Equation 33 shows the simplified calculation for a set of

events that are independent from each other.

$$P(A \cup B) = 1 - P(A^c \cap B^c) \tag{31}$$

$$P(E_1 \cup E_2 \cup \ldots \cup E_n) = 1 - P(E_1^c)P(E_2^c|E_1^c)P(E_3^c|E_1^c \cap E_2^c)\ldots P(E_n^c|E_1^c \cap E_2^c \cap \ldots \cap E_{n-1}^c) \tag{32}$$

$$P(E_1 \cup E_2 \cup \ldots \cup E_i \cup \ldots \cup E_n) = 1 - \prod_{i=1}^{n} P(E_i^c) \tag{33}$$

At this point, every construct is defined that is necessary to perform the calculations. However, the equations are quite complicated and hard to generalize into arbitrary sizes. If the series and parallel structures themselves can be simplified into elemental forms using recursive logic again, any n-long structure can be broken into n-1 recursive calculations for the aggregate probability. The scheme of recursive calculation can be seen graphically in Figure 30 and in equation form in Equations 34 and 35.

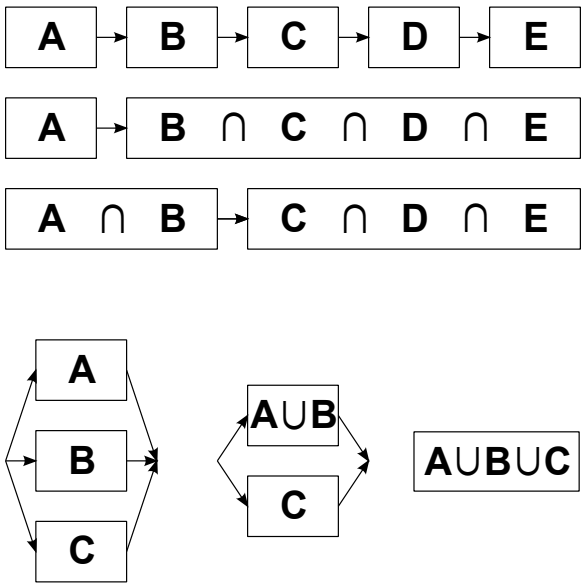


Figure 30: Recursive calculation scheme for series and parallel networks

$$
\begin{aligned}
P(\mathrm{A} \cap \mathrm{B} \cap \mathrm{C} \cap \mathrm{D} \cap \mathrm{E}) &= P(\mathrm{A})P(\mathrm{B} \cap \mathrm{C} \cap \mathrm{D} \cap \mathrm{E}) \\
&= P(\mathrm{A})P(\mathrm{B})P(\mathrm{C} \cap \mathrm{D} \cap \mathrm{E}) \\
&= P(\mathrm{A})P(\mathrm{B})P(\mathrm{C})P(\mathrm{D} \cap \mathrm{E}) \\
&= P(\mathrm{A})P(\mathrm{B})P(\mathrm{C})P(\mathrm{D})P(\mathrm{E}) \qquad (34)
\end{aligned}
$$

$$
\begin{aligned}
P(\mathrm{A} \cup \mathrm{B} \cup \mathrm{C}) &= 1 - (1 - P(\mathrm{A}))(1 - P(\mathrm{B} \cup \mathrm{C})) \\
&= 1 - (1 - P(\mathrm{A}))(1 - (1 - (1 - P(\mathrm{B}))(1 - P(\mathrm{C})))) \\
&= 1 - (1 - P(\mathrm{A}))(1 - P(\mathrm{B}))(1 - P(\mathrm{C})) \qquad (35)
\end{aligned}
$$

Once all the above steps are written in algorithms in a computer executable fashion, the calculations are carried out using mostly recursive structures with little computational power. The structure is separated from equations and the networks to be modeled can be scaled and generalized. The Code Block 5.1 below shows a Python implementation. As can be seen from the code, there is only one conditional function that works entirely recursively. The example inputs A–F belong to the network shown in Figure 31. If root nodes of the network is not known ahead of time, dictionaries for the nodes can be instantiated empty and modified post-initialization to alleviate *name not defined* errors.

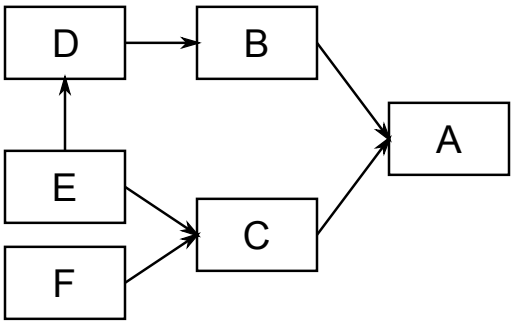


Figure 31: Example probability network corresponding to the Python code

```
1 F = { "p": 0.6 , "In": [] }
```

```
E = { "p": 0.5 , "In": [] }
D = { "p": 0.4 , "In": [E] }
C = { "p": 0.3 , "In": [E, F] }
B = { "p": 0.2 , "In": [D] }
A = { "p": 0.1 , "In": [B, C] }
def prob(X):
    if len(X["In"]) == 0:
        return X["p"]
    elif len(X["In"]) == 1:
        return X["p"] * prob(X["In"][0])
    else:
        neg = 1
        for elem in X["In"]:
            neg = neg * ( 1 - prob(elem) )
        return X["p"] * ( 1 - neg )
```

Listing 5.1: Example network definition in Python

5.4 Deterministic, dynamic, and continuous models

5.4.1 Continuous-time Markov chains

In continuous Markov Chains, a network of nodes and connections represents the states that a system can be in and all the transitions into and out of these states. This representation is somewhat similar to a type of network investigated using graph theory. In fact, all Markov chains are mathematical graphs and the similarity exists even within the development of mathematical rules for solving/simulating a continuous-time Markov Chain model. However, the meaning assigned to model elements is different from the assigned meaning to mathematical graph elements (vertices and edges). Markov chains model the state of a single system, whereas graph theory is used to represent a network of systems.

In Markov Chains, the vertices are called states and the edges are called transitions. The model then starts at a specific state at reference time $t = 0$, usually referred to as the initial state, and progressively transitions out of it towards the final state. A Markov Chain study focuses on the dynamic transition behaviors such as the

time it takes to transition to the final state, the determination of the final state (e.g., convergence, oscillation, chaos), and determination of states that cause bottlenecks. The transitions are modeled as exponential distributions. While, this assumption is a limiting one, it arises from the mathematical formulation and additionally, exponential distributions are the only fitting distributions that can be solved analytically. An example of a continuous Markov Chain is shown in Figure 32.

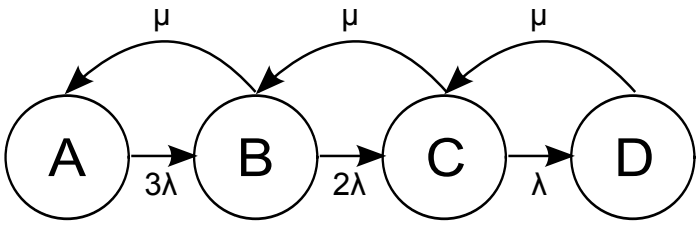


Figure 32: An example continuous Markov chain

In Markov Chains the states include a single piece of information: the probability of the system being in that state at that given time. Therefore, the probability information is only a function of time and time alone. The entire state space of the network can be described in a single list of size n, where n is the number of states (nodes in the network). Each element in this list corresponds to the probability of being in a specific state at time t, and is, therefore, constrained to lie $\in [0, 1]$. The sum of the elements of this list, is constrained to be equal to 1 at all times $t \geq 0$, because the probability of being within the network is always equal to 1 (the network has no entries, exits, sinks, or sources). In other words *the total probability is always conserved* in a Markov chain. Also, all elements of the state list are non-negative (they can be zero or positive). It can be shown that each element has to be a number between 0 and 1, which is expected as they represent a probability. The rules of the state list are given in Equations 36, 37, and 38. Because the state list's order is important, the list is a tuple, which is a mathematical construct that represents an ordered set of elements.

Let $f_i(t)$ be the probability of being in state i at time t. Let i be an integer

between 1 and n with n denoting the number of states in the system. It follows that,

$$\sum_{i=1}^{n} f_i(t) = 1 \tag{36}$$

$$f_i(t) \geq 0 \tag{37}$$

Equation 37 can also be written as

$$0 \leq f_i(t) \leq 1. \tag{38}$$

The list as defined here is not a vector as it does not satisfy some of the axioms a vector has to satisfy. Firstly, a summation of two state lists has no meaning and will violate the sum of elements being equal to 1 rule: if both state vectors' elements sum up to 1, then the sum of the elements of their sum will add up to 2. Additionally, multiplying the state vector by a scalar is nonsensical as well. Any multiplication by other than 1 will result in the sum of the elements being not equal to 1. These violations are shown in Equations 39 and 40.

Let $v(t) = \{f_1(t), f_2(t), \ldots, f_n(t)\}$. It follows that,

$$v(t) + v(t) \text{ is meaningless } \therefore v(t) \text{ is not a vector.} \tag{39}$$

Additionally, for $v(t)$ to be a vector its scalar multiple must remain in the same vector space, i.e., $v(t) \in \mathbb{R}^n \Rightarrow cv(t) \in \mathbb{R}^n$. However, a state list does not follow this property. As shown in Equation 40, $cv(t)$ and $v(t)$ are not in the same vector space; therefore, $v(t)$ is not a vector.

$$c\sum f_i(t) \neq 1 \text{ if } c \neq 1 \therefore cv(t) \text{ is not a state vector.} \tag{40}$$

Unfortunately, tuples are not easily manipulated, and because the state is represented in a tuple, computation is not straightforward. This problem can be alleviated by setting up the problem in an appropriate way where, not by definition but by setup the elements in the state *vector* are always between 0 and 1 and their sum is

always equal to 1. This way the constraints on the state tuple can be removed and it can be defined as a vector. This setup imposes some additional rules in the definition of transitions, but overall the entire mathematical system is easier to deal with.

Transitions between states represent the transfer of probability from current state to a future state, sometimes referred to as a *flux*. Because this is a continuous time Markov Chain, the transitions represent the rate of change of probability of being in the two states they connect, decreasing in one and increasing by the same amount in the other. The amount has to be the same, to keep the sum of elements of the state vector equal to 1. Using this practical trick, the state tuple can be treated as a vector. A generic example is considered in Figure 33 and the corresponding equations are given in Equations 41–44.

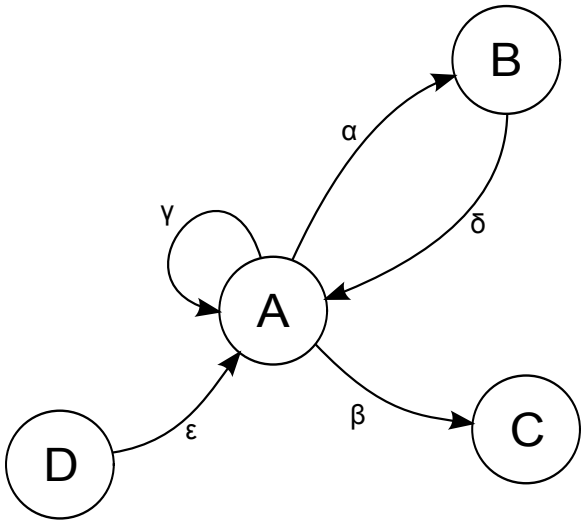


Figure 33: Example Markov chain transitions for a single state

$$g_i(t) = \sum_{j=1}^{k} (\text{From}_j f_j - \text{To}_j f_i) \tag{41}$$

where g is the total transitions of state i, k is the number of states, and f is the probability of being in a given i at time t.

Transition out of A can be abstracted as:

$$g_A(t) = \alpha f_A(t) + \beta f_B(t) + \gamma f_A(t). \tag{42}$$

Transition into A can be abstracted as:

$$h_A(t) = \delta f_B(t) + \epsilon f_D(t) + \gamma f_A(t). \tag{43}$$

The total effect on state A can be found by adding the two fluxes above:

$$\begin{aligned}\frac{\mathrm{d}f_A(t)}{\mathrm{d}t} &= h_A(t) - g_A(t) \\ &= \delta f_B(t) + \epsilon f_D(t) + \cancel{\gamma f_A(t)} - \alpha f_A(t) - \beta f_B(t) - \cancel{\gamma f_A(t)} \\ &= \delta f_B(t) + \epsilon f_D(t) - \alpha f_A(t) - \beta f_B(t). \end{aligned} \tag{44}$$

One can think of the transitioning as a *flow of probability*. Because there can be only one transition from one state to another (if there are more, they can be summed up into a single one), the transitions out of one state can be represented as a list, similar to the state list. In this case, the list is a vector because there are no restrictions on the numbers in this list. The list can be freely manipulated by scaling and addition operations, and in turn satisfies the vector axioms. The vector form of Equation 41 is given in Equation 45.

$$\underline{g_i}(t) = \begin{bmatrix} \mathrm{From}_1 & \mathrm{From}_2 & \cdots & \mathrm{From}_k \end{bmatrix} \begin{Bmatrix} f_1 \\ f_2 \\ \vdots \\ f_k \end{Bmatrix} - \begin{bmatrix} \mathrm{To}_1 & \mathrm{To}_2 & \cdots & \mathrm{To}_k \end{bmatrix} \begin{Bmatrix} f_1 \\ f_1 \\ \vdots \\ f_1 \end{Bmatrix} \tag{45}$$

The transition vectors of all states can be put in a list, which can act as a matrix. The transition matrix holds all the information about the transition rates from any state to every other state. Now, the state vector can be linearly mapped to a future state vector by the transition matrix. This mathematical formulation, which is a system of homogeneous first order linear differential equations, can be solved (simulated) for the dynamic state transition behavior of the system being modeled. The general form of this equation is given in Equation 47. An example of a Markov Chain network and its transition matrix is given in Figure 34.

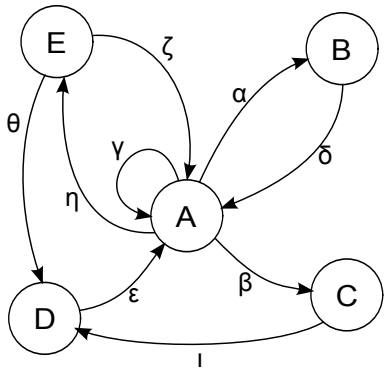


Figure 34: Example Markov chain network

$$\underline{\underline{T}} = \begin{bmatrix} -\alpha-\beta-\eta & \delta & 0 & \epsilon & \zeta \\ \alpha & -\delta & 0 & 0 & 0 \\ \beta & 0 & -\iota & 0 & 0 \\ 0 & 0 & \iota & -\epsilon & \theta \\ \eta & 0 & 0 & 0 & -\theta-\eta \end{bmatrix} \tag{46}$$

$$\mathrm{d}\underline{V}(t) = \underline{\underline{T}}\ \underline{V}(t)\mathrm{d}t \tag{47}$$

In order to calculate the state space at a given time, Equation 47 is integrated from $t = 0$ to $t = t^*$.

$$\underline{V}(t^*) = \int_{t=0}^{t=t^*} \underline{\underline{T}}\ \underline{V}(t)\,\mathrm{d}t + \underline{V}(t=0) \tag{48}$$

The simulation then is transformed into a mathematical evaluation of an integral. In Markov Chains the transitions are a function of only the current state (i.e., the entries of the matrix are constants), which makes this integral analytically solvable. This property is called *memoryless* or *Markov*. The condition is represented by Equation 49 [191]. The general solution technique is given in Equations 50 and 51. It can be seen from the equation that the solutions are indeed exponential distributions. The equation for a cumulative distribution function of generic exponential distribution is given in Equation 52 [191]. Exponential distributions are memoryless [191].

$$P(X \geq x) = P(X \geq x_0 + t \mid X \geq x_0) \tag{49}$$

$$\mathrm{d}\underline{X}(t) = \underline{\underline{A}}\underline{X}(t)\mathrm{d}t \tag{50}$$

$$\underline{X}(t) = e^{\underline{\underline{A}}\underline{X}(t)} + \underline{X}(t=0) \tag{51}$$

$$F(x;\lambda) = \begin{cases} 1 - \exp^{-\lambda x}, & \forall x \geq 0, \\ 0, & \forall x < 0. \end{cases} \tag{52}$$

Once this equation is solved, it can be evaluated for all states and $t \geq 0$. The transition dynamics can be plotted for all states, questions can be asked whether the final state can be achieved in a reasonable amount of time, whether a state sees some build-up (the system is stuck in a state), whether there are oscillations, whether the steady-state ($t \to \infty$) solution is satisfactory, etc.

Because the answer is in probability form, no repetitions are necessary for probabilistic analysis. The answer is the converged solution of infinite trials. This fact makes Markov Chains extremely powerful when dealing with stochastic problems, even though this form of Markov Chains is technically deterministic. As discussed earlier, most systems of systems show stochastic behavior in one form or another. Markov analysis of dynamic behaviors should be considered for all system of systems problems. However, not all problems are reducible to a Markov Chain. Markov/memoryless property is not applicable to all problems practically. Markov Chains are also limited to single elements traveling within a network. If there are multiple elements traveling, their transitions and states have to be independent of each other, and have to be analyzed in separate Markov analyses (this adds to the number of equations to be solved). Markov Chain networks also cannot include *AND* statements as they violate the memoryless property.

5.4.2 System dynamics

System Dynamics is yet another formalism for systems modeling. Originally called industrial dynamics by Forrester [71] , system dynamics is used for large-scale industrial, management, education, policy, social, ecological, and economic analyses [178].

System dynamics borrows elements from Markov chains as well. In system dynamics nomenclature The states in Markov chains are called levels (or stocks) and the transitions are called flows (or rates). Despite the similarities in elements, system dynamics and Markov chain formulations are mathematically and semantically very different.

Instead of having all transitions as exponential distributions, system dynamics allows for other types of functions. Also, system dynamics models allow for the tracking of metrics other than probabilities, such as amounts/counts of things, money, physical measurements, and abstractions such as desirability of reliability. Finally, system dynamics allows for unit changes in the transition so that nodes do not represent states of a system but some properties of it. Each node in the network now represents only a property of the system so that the state of the system is the entire set of property-value tuples.

Semantically, levels do not resemble states. They are containers of some quantity and represent the accumulation of flows [72]. Interestingly, this quantity can be the probability of being in a state just like Markov chain formulation, but it can be another quantity as well. Stocks usually represent measurable quantities such as money [72], number of people [198, 72], and amount of goods [18, 72] or abstract quantities (qualitative elements) such as desirability [126], information [72], and happiness [126]. In system of systems context, they could represent data collected and stored from a reconnaissance mission, munitions transfered and spent, passengers in a given airport, among others. Levels are unknown variables that describe the state the system is currently in [72].

Flows represent influences of levels on other levels. Flows change the values in levels and as such they represent activities withing the system [72]. The quantities in stocks go through flows and arrive at other stocks transformed by the equation included in the flow. These equations are called decision functions. It is important to maintain proper unit conversions for physical quantities. Unlike Markov chains,

flows can make or destroy properties. The flows now represent influences instead of Markovian transitions and as such do not have to conserve the quantities they carry between levels. For example, a flow can multiply the *flowing quantity*, and return it back to the same level, increasing its quantity significantly. Flows transport and modify quantities from the *upstream* level to the *downstream* level.

Mathematically, levels are unknown variables in a system of differential equations. The equations are embedded into the flows and each flow includes two equations: *out of upstream level* and *into the downstream level*. Another element that system dynamics has is a *parameter*. Because all parameters inside flow equations need to be graphically represented, parameter blocks are connected to flows via an information arrow. System dynamics formalism is a graphical one and as such the inclusion of every parameter in equations is key to transparency. Unlike Markov chains, the flow equations in system dynamics are all different; therefore, showing information arrows and parameter blocks is a good practice. An example of a simple system dynamics model that includes levels, flows, parameters, and information arrows is shown in Figure 35 and the related equations are given in Equations 53–55.

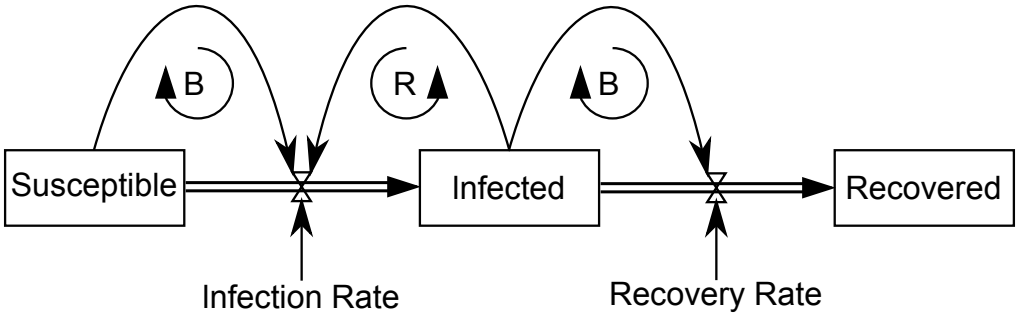


Figure 35: Example system dynamics network that represents an epidemic

Let λ and μ represent infection and recovery rates respectively. Also, let $S(t)$, $I(t)$, and $R(t)$ represent susceptible, infected, and recovered population as a function of time. The system dynamics model shown in Figure 35 can be written as shown below. In three fairly simple integral equations that can be solved easily with numerical

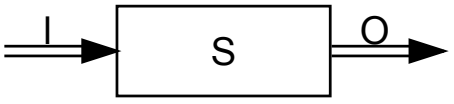
integrators, a disease model with a fixed population and infection rate can be modeled. System dynamics is a powerful method in thinking about systems and representing them in easy to understand constructs.

$$S(t) = -\int_0^t \lambda S(t)I(t)\mathrm{d}t + S(0) \tag{53}$$

$$I(t) = \int_0^t \lambda S(t)I(t)\mathrm{d}t - \int_0^t \mu I(t)\mathrm{d}t + I(0) \tag{54}$$

$$R(t) = \int_0^t \mu I(t)\mathrm{d}t + R(0) \tag{55}$$

Other than these elemental constructs, there are source, sink, and delay elements. Sources and sinks create or destroy quantities that flow out of or into them using flow connectors, respectively. Inclusion of sources and sinks changes the nature of the mathematical equations that govern the dynamics of the systems to non-homogeneous differential equations. The final general equation type that needs to be solved for simulation is then a system of non-homogeneous first-order nonlinear differential equations. The main equation of a system dynamics model is the equation for a stock. This equation is shown in Figure 36. Other equations are added as seen fit by the modeler. Delay elements are the most complicated elements in system dynamics formulations. In their diagram box, they include many variables regarding the order and type of delays [73]. Most frequently used delays are of the exponential and discrete types.



$$S = \int_0^t (I - O)\mathrm{d}t + S_0$$

Figure 36: The equation for a level with an input and an output flow

The real power of system dynamics simulations is realized when the networks include a certain number of feedback structures. These feedbacks are called *reinforcing* if their effect is to enhance the change of quantities in a level, and *balancing* if their effect is to work against the change of quantities in a level. This formulation is similar

to static stability/instability. Examples of reinforcing and balancing loops were given in Figure 35. In the example, the more the number of infected people, the faster the infections occur; however, the more the number of infected people, the more recoveries happen. The first one is a reinforcing loop, the second is a balancing loop.

The non-homogeneous nonlinear first-order ordinary differential equations are usually not analytically solvable. These equations now need to be solved numerically and depending on the method used, this step may reduce the accuracy and create convergence issues. Today, reliable numerical ordinary differential equations solvers exist for most software packages, which alleviates the problem for reasonably sized networks with fairly short simulation times. As the simulation duration is increased, the numerically solved equations tend to build up errors.

It is important to mention that when it was invented, system dynamics was not entirely a new concept. It is a graphical representation of ordinary differential equations. It emphasized form over semantics and therefore is a formalism. Human mind can grasp the influences of parameters on other parameters better graphically than symbolically. Larger models can be constructed easier with graphical building blocks, rather than inserting symbolic mathematical terms into systems of already complicated equations. A number of software packages with graphical user interfaces for system dynamics modeling exist commercially and freely [83, 107, 162, 187, 197, 200].

5.5 Deterministic, dynamic, and discrete models

5.5.1 Discrete-time Markov chains

Similar to continuous Markov chains, discrete Markov chains consist of two elements: states and transitions. These models have limiting assumptions, but they can uncover oscillatory or stable behavior as time tends to infinity with fairly low computational expense. Because a discussion of Markov chains fundamentals was given in Section 5.4.1, the focus of this section will be more on the changes due to the

discreteness.

In discrete Markov chains, the transitions are instantaneous and happen during discrete time steps. Imagine a discrete Markov chain that has a step time of 2 seconds as an example. At time $t = 4$, the Markov chain model has some numbers in its states (vertices), and at the next time $t = 6$, it will have some numbers (possibly different numbers) in its states again. However, it does not have any state at time $t = 5$, as such an existence is not defined by the model. Discreteness applies only to time, the numbers in states are not discrete. Just like continuous Markov chains, the numbers assigned to the states are analogous to the probability of the system to be in that state at a given time. The numbers in the states can only assume values between 0 and 1, and the sum of all the values in the states still add up to 1. The states can be gathered into a list, or using the same trick described for continuous-time Markov chains (on page 115) into a vector. The transitions do have a different meaning compared to the continuous case. In the discrete Markov chains the number attached to the transition represents the probability of that transition happening at each time step. In the continuous case, this number was the rate parameter in the exponential distribution. An example network is given in Figure 37. On the same figure, the evolution of the Markov chain model in three time steps is depicted.

Unfortunately, discrete-time Markov chains are not solved as functions that can simply be plotted. However, the equations for the transition have desirable qualities. A very brief discussion on discrete-time Markov chain mathematics is given below.

State vectors can be represented by column or row vectors (in this work a column vector form is used). The definition is given in Equation 56. Given a state vector, the next state vector can be found by multiplying it by the transition matrix as shown in Equation 58. The state vector for the example Markov chain depicted in Figure 37 at each time step is given in Equation 57. The transition matrix of the same Markov chain is given in Equation 59. It is important to note that the transition matrix is a

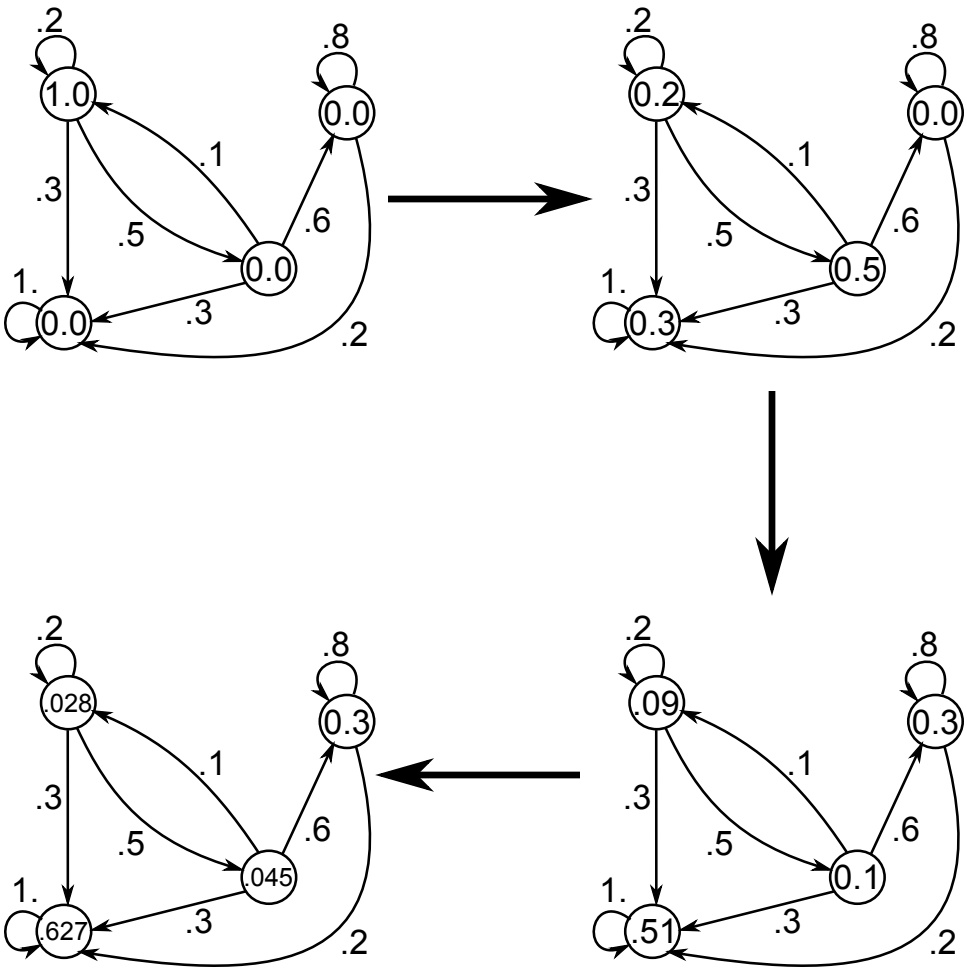


Figure 37: Discrete-time Markov chain transitions

constant matrix and it does not change with respect to time.

State vectors can be obtained by repetitive multiplication of the transition matrix. Equation 60 shows a general formula for obtaining a state vector for a given time step. There are some decomposition techniques for a square, non-negative matrix that can help with the calculation of the powers of a transition matrix.

$$\underline{V}(t_i) = \begin{Bmatrix} P(\text{system in state 1 at time step } i) \\ P(\text{system in state 2 at time step } i) \\ \vdots \\ P(\text{system in state n at time step } i) \end{Bmatrix} \tag{56}$$

$$\underline{V}(t_0) = \begin{Bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{Bmatrix}, \quad \underline{V}(t_1) = \begin{Bmatrix} 0.2 \\ 0.5 \\ 0.3 \\ 0 \end{Bmatrix}, \quad \underline{V}(t_2) = \begin{Bmatrix} 0.09 \\ 0.1 \\ 0.51 \\ 0.3 \end{Bmatrix}, \quad \underline{V}(t_3) = \begin{Bmatrix} 0.028 \\ 0.045 \\ 0.3 \\ 0.627 \end{Bmatrix} \tag{57}$$

$$\underline{V}(t_{i+1}) = \underline{\underline{T}}\, \underline{V}(t_i) \tag{58}$$

$$\underline{\underline{T}} = \begin{bmatrix} 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0 & 0 & 0 \\ 0.3 & 0.3 & 1 & 0.2 \\ 0 & 0.6 & 0 & 0.8 \end{bmatrix} \tag{59}$$

$$\underline{V}(t_i) = \underline{\underline{T}}\, \underline{V}(t_{i-1}) = \underline{\underline{T}}\, (\underline{\underline{T}}\, \underline{V}(t_{i-2})) = \cdots = \underline{\underline{T}}^i \underline{V}(t_0) \tag{60}$$

Perhaps more importantly, a steady-state analysis can be performed using the Perron-Frobenius Theorem [30]. There is a unique state vector that satisfies the condition given in Equation 61. Any eigenvalue other than $\lambda = 1$ is not a valid eigenvalue (the sum of all elements in the state vector must equal 1). This unique state vector is the steady-state solution to the discrete-time Markov chain. Using this technique, discrete-time Markov chains can be solved for large values of time steps very efficiently. For the example given in Figure 37, it can be seen that the third state (bottom left), has no exit. This is a clue that all probabilities will gather in that state eventually. In fact, the eigenvector corresponding to the eigenvalue that is equal to unity, represents exactly that. This eigenvector is given in Equation 62.

$$\underline{V}(t_{i+1}) = 1\underline{V}(t_i) = \underline{\underline{T}}\underline{V}(t_i) \tag{61}$$

$$\lim_{i \to \infty} \underline{V}(t_i) = \begin{Bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{Bmatrix} \tag{62}$$

In addition to solving Markov chains analytically, given a starting state, a Markov chain model can be executed stochastically (i.e., every execution results in a different outcome). Such Markov chains will be discussed under stochastic models. Even though deterministic Markov chains model stochastic systems, as analytical models they are deterministic. For example, given a transition matrix such as the one above, the steady-state solution is always that given in Equation 62.

5.5.2 Petri nets

Petri nets are similar to discrete-time Markov chains with one major difference: Petri nets describe the states of multiple systems. Petri nets have found many applications in fields such as reliability [188], communications [32], manufacturing [164], chemistry [165] and many more. Petri nets are formulated somewhat differently compared to Markov chains but they are still mathematical graphs. The discussion below describes a Petri net.

Petri nets are directed, bi-partite graphs. A bipartite graph's vertices can be decomposed into two disjoint sets such that no two vertices within the same set are adjacent [194]. Figure 39 depicts a bipartition. There are two types of vertices: a *place* and a *transition*. Transitions in a Petri net are represented as vertices (contrast this to Markov chain transitions). A bi-partite graph is a graph, in which vertices can be organized into two disjoint independent sets and there are no intra-set connections (i.e., a vertex in one set can only connect to a vertex in the other set via an edge). Figures 38 and 39 show a Petri net example and its bi-partite separation.

Petri net places hold a number of *tokens*, depicted as dots within places. These tokens can be colored differently to represent different kinds of systems (colored Petri net). If multiple places are leading to a transition, in order for that transition to *fire*, all places have to be occupied. If a transition leads to multiple places, all the places get a number of tokens when it fires. Figure 40 represents a simplified Petri net model

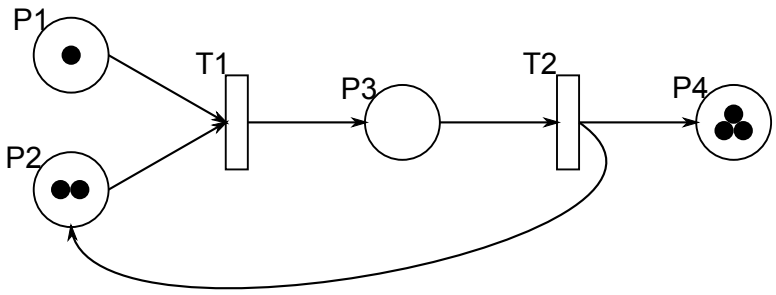


Figure 38: An example Petri net graph

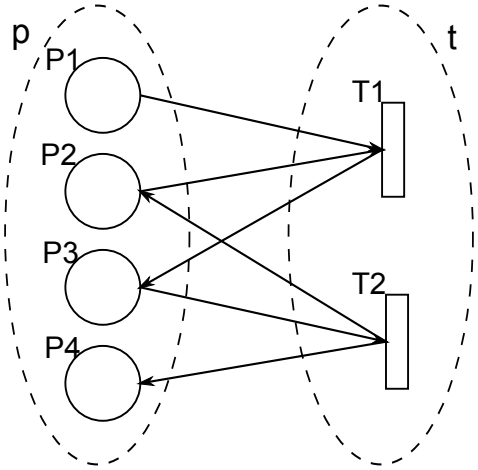


Figure 39: Example Petri net graph in bi-partite arrangement

for the combustion of methane. In this model, if there was only one O_2 molecule in the beginning, the *react* transition would never fire, and methane would not burn in the simulation.

Almost all Petri net models are stochastic models. However, some special cases can be simulated as deterministic models. The main source of stochasticity stems from the fact that sometimes places are connected to multiple transitions and if they happen to fire simultaneously, the simulation would choose one at random to fire and cancel the other. Such simultaneous firings, conflicts, and confusions are dealt with within the simulation engine. Stochastic Petri nets will be dealt with under stochastic models. However for the purposes of this section, it is important to note that not all Petri net graphs can be simulated as a deterministic model.

Petri net models create and destroy tokens, unlike Markov chains where the total

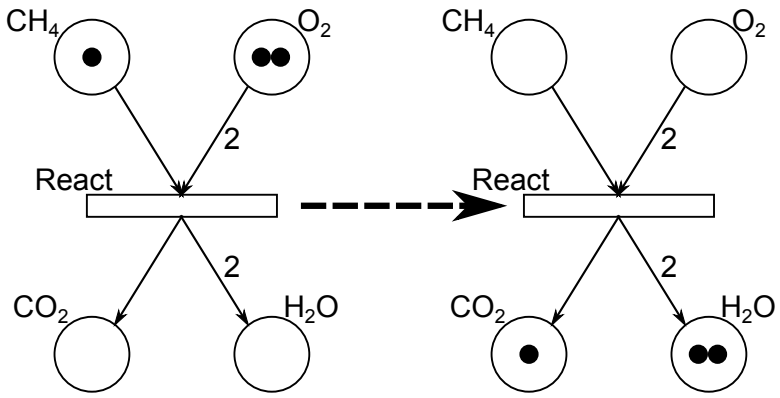


Figure 40: A Petri net model representing methane burning

probability is always equal to unity. Such properties and the ability to model multiple entities are useful properties of Petri nets for system of systems modeling. Activity diagrams used in UML and SysML can be translated into Petri net formalism. DoDAF does not provide a standard in creating its viewpoints and is more flexible than the other two modeling description languages. DoDAF's OV-5b and SV-10b are likely candidates for a Petri net formalism and model based on their descriptions [59].

5.6 Stochastic and static models

5.6.1 Monte Carlo

Monte Carlo is the general name given to random sampling for simulations. When analytical deterministic solutions are not readily available for stochastic problems, multiple samples of the stochastic scenario can be executed and the results analyzed statistically. In order for Monte Carlo analysis to work large numbers of samples are needed. Due to this reason, Monte Carlo method is only practical for the solution of stochastic problems with a fast execution time.

Figure 41 shows the integral of e^{-x} between 0 and 1 using a Monte Carlo simulation. Random points are generated inside $(0,0),(0,1),(1,1),(1,0)$. If the y value of these points are less than the e^{-x} they are counted, otherwise they are discarded. The total count is later divided by the number of points used in the simulation and multiplied by the area that the random points were generated in (in this particular

case that area is equal to 1). The example in Figure 41 used a thousand points and estimated the area under the curve to be 0.635 (the real answer is roughly 0.632).

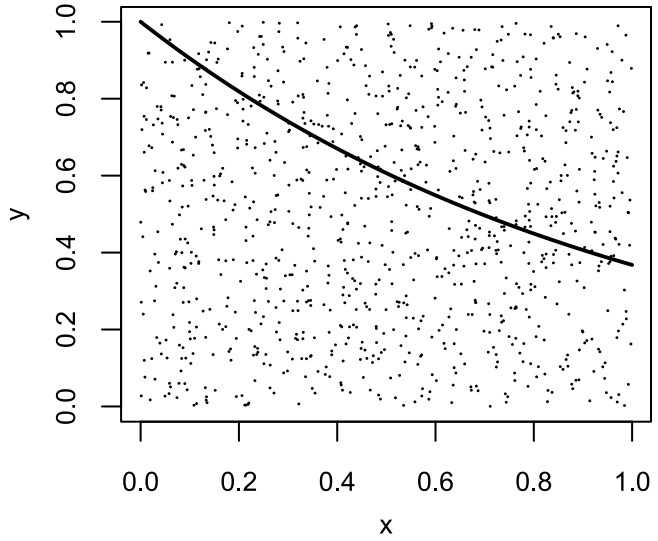


Figure 41: Monte Carlo simulation for the integral of e^{-x} between 0 and 1

Monte Carlo simulations are usually used on top of other modeling methods. For example, a stochastic Petri net model can be run repeatedly using a Monte Carlo simulation. By themselves, they are of limited value, but paired with pseudo-random number generators and other models, Monte Carlo simulations can be invaluable.

5.7 Stochastic, dynamic, and continuous models

Continuous random numbers are best handled analytically. Any Monte Carlo simulations using continuous random numbers are implemented in a discrete setting. For example, a Markov chain simulated via a Monte Carlo simulation can have continuous time, but state changes will end up being discrete steps (only one state will be active in a given time). There are no practical stochastic, dynamic, and continuous models relevant to system of systems analysis.

5.8 Stochastic, dynamic, and discrete models

5.8.1 Monte Carlo discrete time Markov chains

Markov chains that advance in discrete steps can be simulated case by case using a Monte Carlo simulation. In this formulation, the elements of the *state vector* of the system can be thought to equal to exactly zero, except for one element, which will be equal to one. Every transition is definite: if there are multiple possible transitions out of a state, only one will be picked at every time step. In this setting Monte Carlo Markov chains are very similar to Petri nets but still represent a single system.

Monte Carlo Markov chains do not operate on the state vector with a transition matrix. At each step, the simulation looks at the possible transitions out of the state the system is in. Then, it creates a cumulative distribution function based on the choices and their inherent probabilities. Figure 42 depicts such a simulation step. If the system is in the top-left state initially, at the next step, it will be in bottom-left state with a probability of 0.3, in the middle state with a probability of 0.5, and remain in the top-left state with a probability of 0.2.

Monte Carlo Markov chains, therefore, advance steps algorithmically, not analytically. In order to get the probabilities of being in certain states, one has to run the model numerous times. This is not an issue if one is trying to find the steady-state solution, i.e., $t \to \infty$. Simple Markov chains converge fairly quickly. However, in order to get a metric such as *the time to reach the end state with a 90% probability*, the modeler has to perform a large number of runs, execute each run for a large number of steps, save the states of each run and step, and finally average them across the runs. The data input-output and final statistical analysis can be excessively computer intensive. Therefore, Monte Carlo methods should only be used with Markov chains if analytical methods fail for some reason. Harnessing the desirable analytical properties of Markov chains is important in a simulation study.

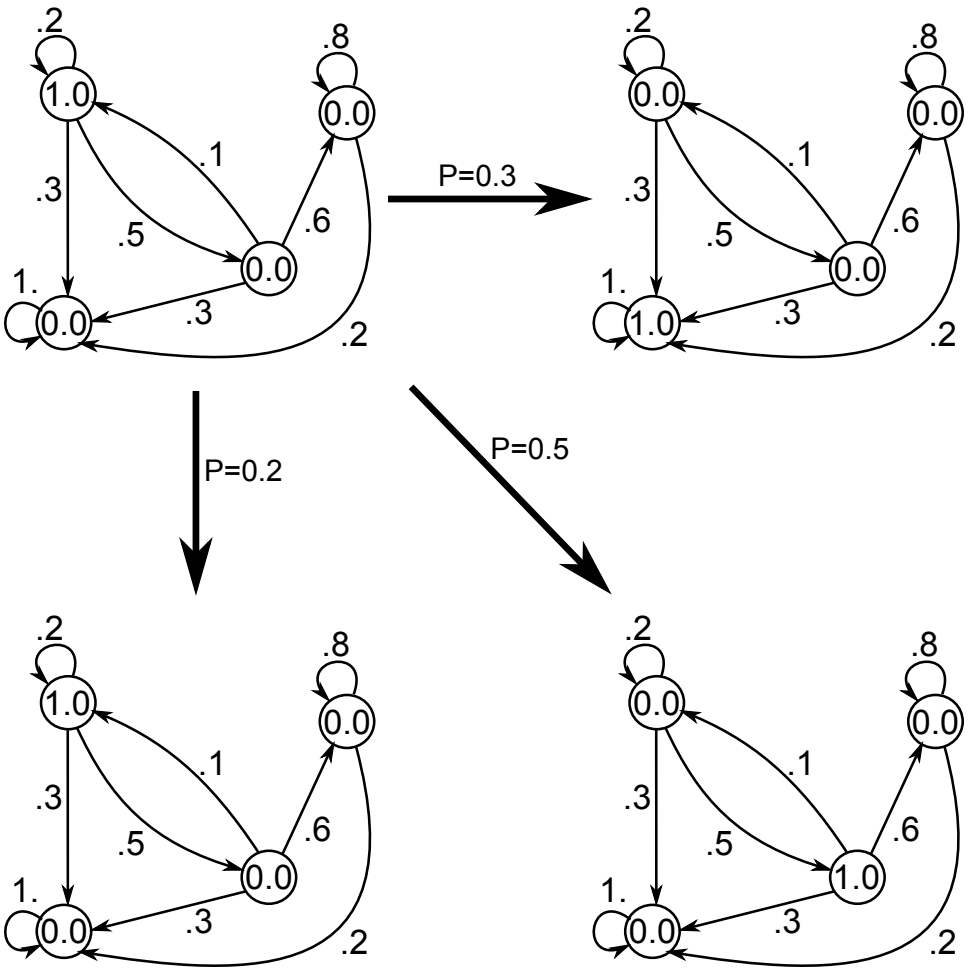


Figure 42: Monte Carlo simulation step for a discrete time Markov chain

5.8.2 Petri nets

Most Petri nets are stochastic models as discussed on page 127. Here, Petri nets that are not carefully crafted to be deterministic are discussed. As per the discussion before there are multiple sources of stochasticity for Petri nets:

1. given a state of the system, multiple transitions are ready to fire simultaneously, but only one is allowed in a single simulation step, or

2. the transitions are not deterministic, they happen with a certain probability or they take a random amount of time to fire, or

3. token aging is enabled in the Petri net.

At each step of the simulation, only one transition can fire. In situations where multiple transitions that share some input places are ready to fire, the order of the firing makes a difference in the simulation output. In such situations, the simulation algorithm must resolve concurrency, conflict, and confusion instances [146] and decide on which transition to fire. In order to obtain unbiased simulation results, such situations are usually randomized and the Monte Carlo method is used to simulate the model.

The second case of random firing resembles a Markov chain transition. Once a transition is enabled (i.e., all of its input places have the necessary amount of tokens), the simulation algorithm either decides whether the transition fires during the next time step or when the transition will fire. The first case is a time-stepped solution. The second case resembles a discrete event simulation. Discrete event simulation algorithm will be discussed later in this section.

Finally, in the third case, some Petri nets include an aging token logic [188]. Token age can affect the firing of transitions in many ways such as changing the schedule, probability, or inhibiting it by removing an older than allowed token. Such a logic would not be admissible in a Markov chain (*memoryless* property would be violated). The use of aging tokens is also technically not part of the standard Petri net formulation, but Petri nets are more easily extensible.

5.8.3 Queueing theory

Queueing models borrow from Markov chains as well as Petri nets. They model a process multiple entities go through. Examples of such processes include communication networks (data packets travel through servers), vehicle assembly lines (a vehicle chasis travels along a line and parts are attached to it throughout), airports (passengers arrive, check-in, go through security, wait at the gate, and board the aircraft), and a Ph.D. process (students enroll in the program, take the qualifying exams, propose a

topic, and defend their thesis someday). Figure 43 shows a simple queueing model.

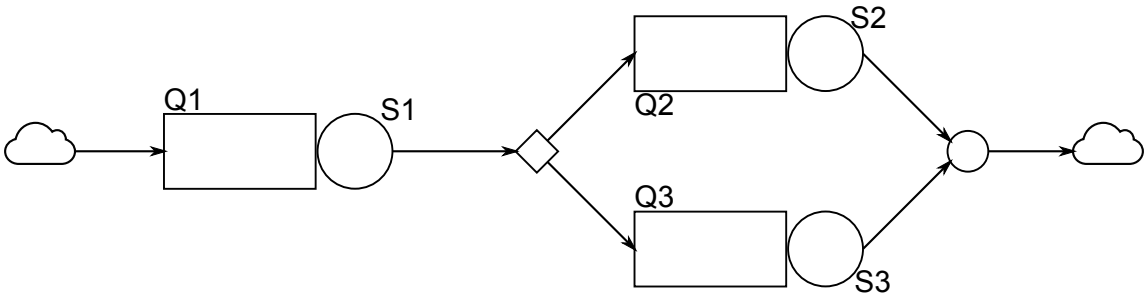


Figure 43: Example queueing model

Queueing models introduce the aspect of a queue ahead of servers, queueing theory counterpart of places, and depending on the model formulation queues can have capacities and rules such as *first in first out* or *priority* processing. Queues are one of the main focuses of queueing models and they are constantly monitored for mean waiting times, mean queue lengths, and instances when they fill up. Not every server must have a queue (i.e., server has infinite capacity), but each queue must have a server. If a queue overfills, the system fails to process the lost customers (they disappear).

Servers process customers in a process time assigned to them. They also have a limit on how many customers they can process simultaneously. If a server is busy, customers arriving at this server's station must wait in its queue. It can be shown that if the arrival rate to the server is faster than the server's processing time, the queue will fill to infinity. However, in the opposite scenario of the arrival rates being slower than the process rates, the server is underutilized. One of the important uses of queueing theory is to optimize a network so that the queues are small but servers are well-utilized.

Each server-queue pair can be characterized by Kendall's notation: A/S/c [114]. A stands for arrival, S stands for service time, and c stands for server capacity. A and S are usually random and drawn from various random populations. However, deterministic arrival and service times with careful scheduling can result in a deterministic

simulation.

5.8.4 Discrete event

Discrete event models are much more flexible compared to the ones described before. Discrete event models are executed to represent the instantaneous changes in system states [121]. The models can be executed using pen and paper as the logic is fairly easy to follow; however, modern systems have too many elements and functions to execute their models without a computer [121]. Discrete event simulation is not suitable for continuous processes such as dynamics (Newton/Euler's Laws), flows, and thermodynamics where first or subsequent derivatives are key analysis assets. For example, the fall of an object is described in Galileo's Gedankenexperiment discussed on page 66. A discrete event model of this system would not be extremely useful.

State 0 The stone is held at some height h above the ground.

State 1 The stone is released and is freely falling.

State 2 The stone is on the ground, resting.

Because the states change instantaneously at a given time, the trajectory of the stone is impossible to calculate using a discrete event simulation. The information on velocity and acceleration is completely lost. Similarly, a discrete event model of an aircraft flying would not be a suitable model for performance calculations such as range, endurance, and turn radius.

The state changes in a discrete event model are done via *events*. A common confusion is to think that discrete event models are discrete in time: they are actually discrete in events, not in time. In most discrete event simulation programs [121], time is continuous (i.e., it does not have fixed-steps) and at specific times discrete events (i.e., instantaneous state changes) happen. These events are comparable to transitions in time discrete Markov chains and Petri nets.

In discrete event models, the network of elements represent a large-scale process. Again, this network can be represented as a graph. The elements of this graph are *events* (sometimes referred to as *processes*), *transitions*, and *queues*. Discrete event models are very similar to queueing models in this respect. Additional logical elements can be added to discrete event models such as *joiners*, *splitters*, *and/or gates*, etc. All queueing theory elements are included in discrete event models such as queues (including their rules), server capacities, and customers (called entities in discrete event nomenclature). Additionally, discrete event models can operate on the entities and modify them. Entities can have effects on servers, capacities can change over time, etc. Because discrete event paradigm is extremely flexible not every aspect of discrete event simulations will be discussed here.

A discrete event simulation has a number of components:

Main program: This program runs the simulation from the initial conditions to the exit conditions.

Simulation clock: The simulated time is tracked by this variable. The simulation clock is different from the real time and the CPU time.

Events list: This list holds all events for the simulated system. The list is accessed whenever a calculation is required that is related to a specific event.

Entities list: This list holds all entities currently in the system. When information about a specific entity is required, that entity is found within this list.

Future event list: This list contains all the upcoming events and their schedule. The study of various data structures for fast manipulation of this list is an ongoing effort.

Counters: The simulation code is instrumented in specific locations to gather statistics of important simulation metrics.

Event routine: Each event may have a different attribute, logic, sub-calculations. The routine of each event is contained in a sub-program that is repeatedly called by the main program.

Timing routine: Determines the next event and performs future event list operations.

Analyzer routine After the successful execution of the simulation, the analyzer routine processes counters and returns desired statistics for the modeler to investigate.

Library routines: These routines are specific to the programming language used to code the simulation engine. Examples of such libraries include mathematical objects (e.g., trigonometric functions, constants, conversions), pseudo-random number generators, input-output modules, plotting tools, and other useful general programming libraries.

Every discrete event simulator is coded somewhat differently (e.g., routines can be split into sub-routines, lists can be instantiated using different data structures); however, in almost all discrete event simulators, the above list is a minimum set of required software elements. Figure 44 shows a model of a simple discrete event simulator. Most of the computation happens within event routines.

If the event routines are simple but are called a large number of times, list manipulation within the timing routine becomes the driver of the run time. Because the timing routine modifies the future event list (searching, removing, and adding elements to lists are expensive computation procedures) it is important to keep the future event list small. However, if there are a large number of entities in the system, the future event list's growth cannot be avoided. In such cases, it is more desirable to use more efficient data structures such as calendar queues [40].

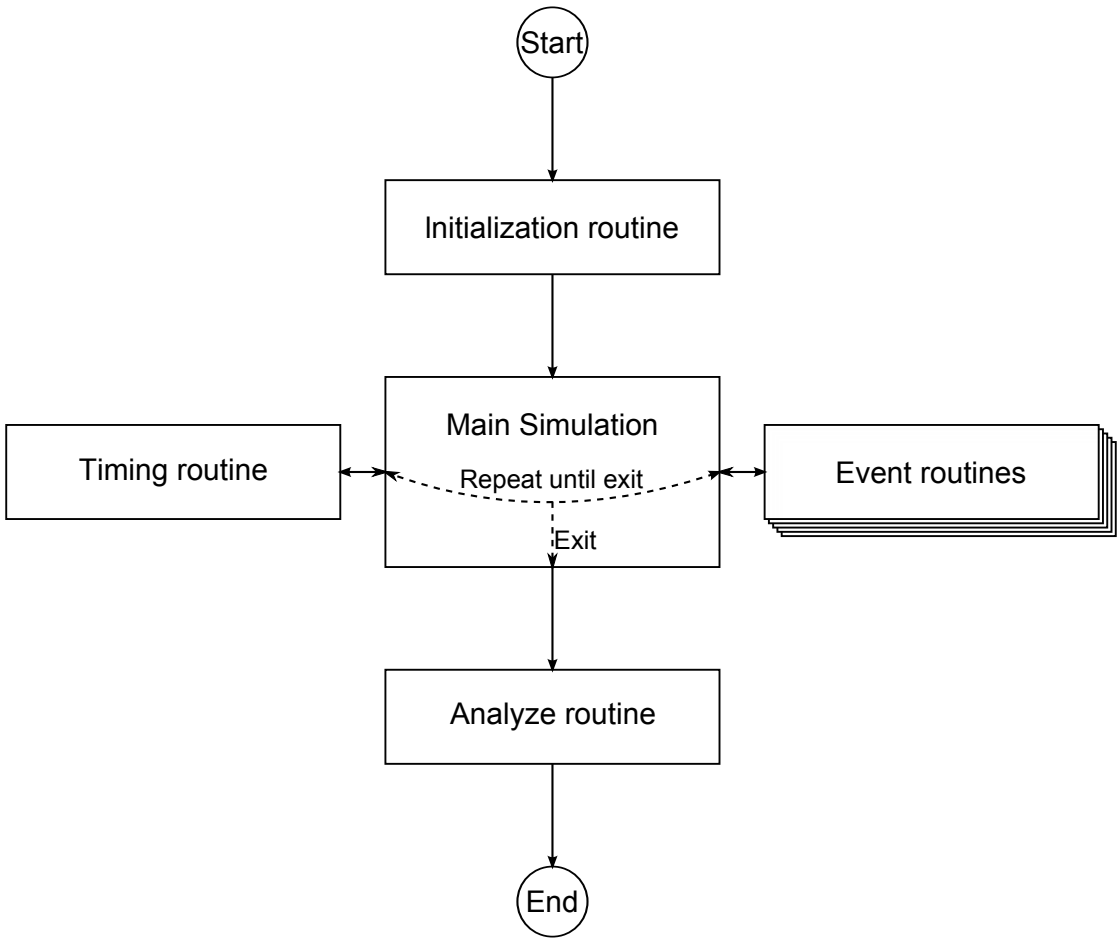


Figure 44: A block diagram of a simple discrete event simulator

5.8.5 Agent-based

Agent-based models are even more flexible than discrete event models. However, this flexibility takes a toll on CPU time as agent-based model simulation algorithms include a large number of nested loops. Agent-based models are used to study the dynamic interactions between the elements of a system and the higher level results of those interaction in the pure bottom-up modeling fashion. Agent-based simulations are one of the best analysis tools for emergent and chaotic systems.

Agent-based simulations are tailored towards a large number of elements. These elements (agents in agent-based nomenclature) constantly monitor their immediate

environments and perform certain actions based on their goals. The agents (representing systems) are coded as autonomous discrete decision-making entities that communicate with each other but make their own decisions [36, 127]. Agents observe their environment, interpret sensory input, analyze options, and make decisions based on their goals. The decisions can be crisp or stochastic (given the same situation, agents can make different decisions), but more importantly, agents can modify their beliefs and goals; therefore, faced with the same situation around them, they can make entirely different decisions to reach their new goal. Agents can be programmed to follow a small number of fairly simple rules or wired to learn through experience via a neural network formulation. Complex emergent behaviors can be obtained even from the simplest rules [36, 127].

Individual runs of an agent-based model can be time consuming (ranging from minutes per run to hours per run, depending on the size and complexity of the model); therefore, it quickly becomes prohibitive to study large numbers of variables within the agent-based model, even when using a well chosen design-of-experiments. Inside the simulation algorithm, the simulation time is stepped. These time steps are of equal length and all simulated phenomena happen at these discrete time indices. In between time steps, the simulation is frozen and nothing is changing. At the time steps every agent performs certain actions based on their own decisions. This is a nested loop: actions loop inside the agents, agents loop inside the simulation loop. Depending on the activity of agents, number of agents, and number of time steps, an agent-based simulation can run for a very long time. Additionally, some agent-based simulation algorithms create links between agents, and these elements need to be tracked and taken into consideration as well.

Even though they are algorithmically not very efficient, they are very useful in cases when the high-level order is not known, but the behaviors of smaller elements can be obtained easily. The modeler must be very careful, however, as small deviations

can cause large changes in high-level behavior of the system as a whole. Also, some systems do not have autonomous elements that make their own decisions. In such cases, agent-based modeling can be thought of as an overkill. Because agent-based models are less mathematical and more algorithmic, efforts to verify and validate them are significantly greater. Each agent-to-agent interaction, agent-to-environment interaction, intra-agent logic, intra-environment logic, and the entire model must be validated in order gain confidence on the model.

Object oriented programming is suitable for agent-based modeling [166]. Because agents are self-contained elements, using an object oriented programming approach is simply easier and requires fewer lines of coding. Also, in the object oriented approach code snippets are reused as many times as possible using *inheritance* and *polymorphism* properties of the approach. Also, agents' attributes are *encapsulated* within the object code, which enables natural development of agent-based models.

Agent-based models do not have a common formalism found in other modeling paradigms. However, they can be developed using regular UML or SysML diagrams just like any other software or system respectively. Figure 45 shows the interactions of an agent with other agents and the environment.

5.9 Transition to experiments and technical work

The following chapters focus on the experimental setup and performed, results obtained, and details about the technical work. The experimental setup is given in the next two chapters and the experimental execution and results are detailed in Chapter 8.

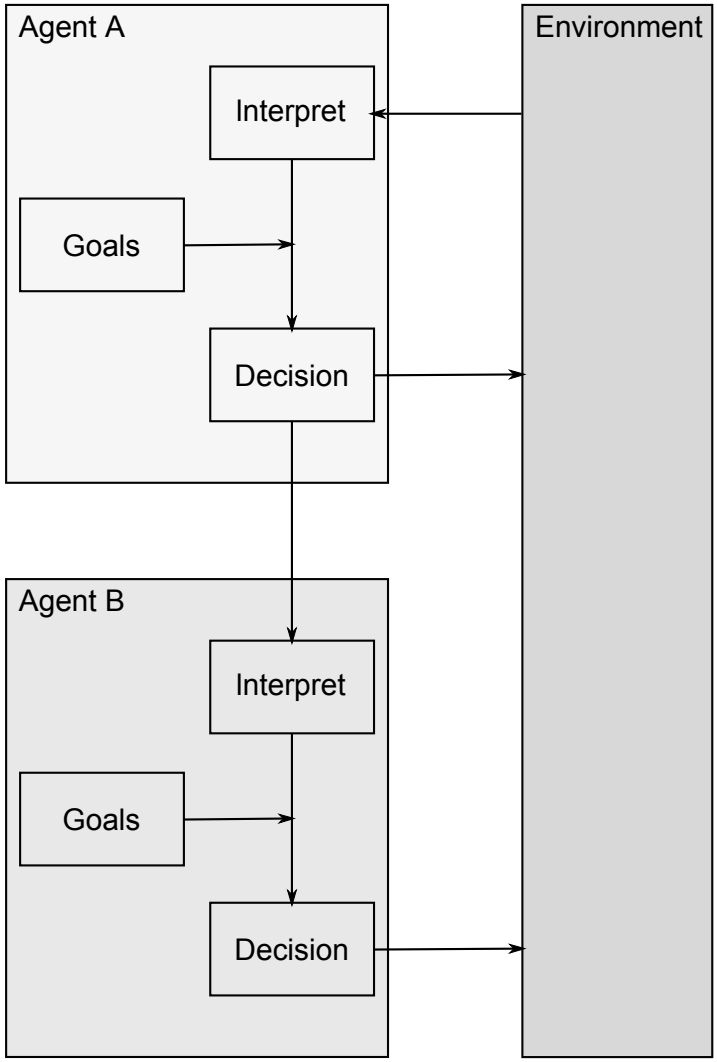


Figure 45: A simplified schematic of an agent-based model

CHAPTER VI

MODELING POTENTIAL OF OPERATIONAL ARCHITECTURE VIEWS

> What we see of the real world is not the unvarnished real world but a model of the real world, constructed so it is useful for dealing with the real world.
>
> Richard Dawkins

One of the main positions taken in this work is that architectures are conceptual models, i.e., abstractions of reality that include enough information to be numerically modeled, solved, simulated, or analyzed. This argument predicts that elements that exist in architecture definitions have computer modeling counterparts. For example an operational node which is part of an OV-2 Operational Resource Flow Description of a system of systems can be reinterpreted as a state which is a part in a possible Markov chain representation of the system of system. A similar process exists for example for Euler's Laws and free body diagrams. A number of equations are written for each moving element in a free body diagram based on the degrees of freedom that element has. There is a direct connection between a free body diagram element and an equation within the system of second order ordinary differential equations used to solve the problem. This chapter and the next investigate such connections between architectural elements and modeling elements.

Before diving into architecture views, a rough estimate of what that work will entail is presented here. There are a large number of DoDAF views that need to be analyzed from a modeler's perspective. Some simplification is therefore necessary. Some views, such as the capability view, for example, can be removed from analysis

mainly because they are not meant to describe the system of system's operation. Additionally, the services view is very similar to the systems views and the analysis of one can be applied to the other without large modifications. Based on these arguments, the analysis here will be limited to the operational (OVs) and systems views (SVs). And even when only OVs and SVs are included, there are still 22 standard views[1] to be analyzed.

Considering the hypothesis set forth in Chapter 3 (a sufficiently complex system of systems will require more than one modeling technique for analysis) these 22 views need to be grouped in feasible but different combinations for each modeling type considered. The problem becomes very large for a human to solve. Specifically, for each model type, every combination of all sizes up to the number of architecture views analyzed must be considered. The calculation is given in Equation 63, where n_a is the number of analyses, n_m is the number of model types, and n_v is the number of views.

$$n_a = n_m \sum_{k=1}^{n_v} \binom{n_v}{k} = n_m \sum_{k=1}^{n_m} \frac{n_v!}{k!\,(n_v - k)!} = 8 \sum_{k=1}^{22} \frac{22!}{k!\,(22 - k)!} = 33,554,424 \quad (63)$$

Each one of the 34 million experiments will require the creation of an architecture consisting of only the combination of views specified in the experiment, and every modeling type must be checked against that combination. The process is meant for a human and cannot be automated. It is evident that this is not a feasible approach to the problem.

Interestingly, in order to find a more feasible solution, the problem must be made larger first. Because it is not obvious what view holds what valuable information for each modeling type, both views and models need to be broken into more definitive constituents. These constituents will be named architecture elements and modeling elements. The problem becomes larger because instead of 22 views and 8 models, there

[1]OV-1, OV-2, OV-3, OV-4, OV-5a, OV-5b, OV-6a, OV-6b, OV-6c, SV-1, SV-2, SV-3, SV-4, SV-5a, SV-5b, SV-6, SV-7, SV-8, SV-9, SV-10a, SV-10b, and SV-10c.

are 50 architecture elements and 24 modeling elements that need to be analyzed. The list of the elements are given in Tables 1–3. The architecture elements are specified in DoDAF specification, and the modeling elements originate from each modeling method's definitions.

Table 1: Operational view elements

View #	View Name	Type	Elements
OV-1	High-Level Operational Concept Graphic	Pictorial	Systems, Actions, Facilities
OV-2	Operational Resource Flow Description	Structural	Operational nodes, Needlines
OV-3	Operational Resource Flow Matrix	Tabular	Activity[a], Info element
OV-4	Organizational Relationships Chart	Structural	Organization, Relationship
OV-5a	Operational Activity Decomposition Tree	Taxonomy	Activity, Relationship
OV-5b	Operational Activity Model	Behavioral[b]	Activity, Misc.[c], Input/Output
OV-6a	Operational Rules Model	Behavioral	Activity, Rules, Relationship
OV-6b	State Transition Description	Behavioral	State, Activity, Transition
OV-6c	Event-Trace Description	Behavioral	Activity, Event, Timeline

[a] Source or sink
[b] Author's interpretation. DoDAF specification document does not offer a classification.
[c] Cost, performer, etc.

The advantage of this approach lies in the reuse of analysis parts. Once an architecture element is found to be useful in being represented as a modeling element, that information can be reused for every view-combination to model mapping. The combinatorial nature of the problem is reduced to a multiplicative one. Equation 64 shows how many mappings are needed, where n_a is the number of analyses, n_{me} is the number of modeling elements, and n_{ae} is the number of architecture elements.

$$n_a = n_{me} n_{ae} = 24 \times 50 = 1,200 \tag{64}$$

Table 2: System view elements

View #	View Name	Type	Elements
SV-1	Systems Interface Description	Structural	System, Sys. Node, Interface
SV-2	Systems Resource Flow Description	Structural	System, Flowline, Port
SV-3	Systems-Systems Matrix	Mapping	System, Resource
SV-4	Systems Functionality Description	Behavioral	System Function, Function I/O
SV-5a	Operational Activity to Systems Function Traceability Matrix	Mapping	System Function, Activity
SV-5b	Operational Activity to Systems Traceability Matrix	Mapping	System, Activity
SV-6	Systems Resource Flow Matrix	Tabular	System[a], Resource
SV-7	Systems Measures Matrix	Tabular	Metric
SV-8	Systems Evolution Description	Timeline	Milestone, Time
SV-9	Systems Technology & Skills Forecast	Tabular	System, Forecast
SV-10a	Systems Rules Model	Behavioral	Logic/Rules
SV-10b	Systems State Transition Description	Behavioral	Function state, System[b]
SV-10c	Systems Event-Trace Description	Behavioral	System[b], Functions, Interactions[c]

[a] Source or sink
[b] Could be a performer, interface, or organization
[c] Information or material exchange

The way the elements are chosen can vary from analyst to analyst; however, the main points should remain the same, e.g., there is a possibility to merge the server and size elements in a queueing model, but the implications of doing so are insignificant as long as the aspect is captured somehow. The analysis in this thesis will be carried with the above selection and stay consistent throughout. Each of the 1,200 analyses requires only an example of how an architectural element could be translated into a modeling element. The following sections will organize the analysis into views and modeling types and discuss the matches between their elements.

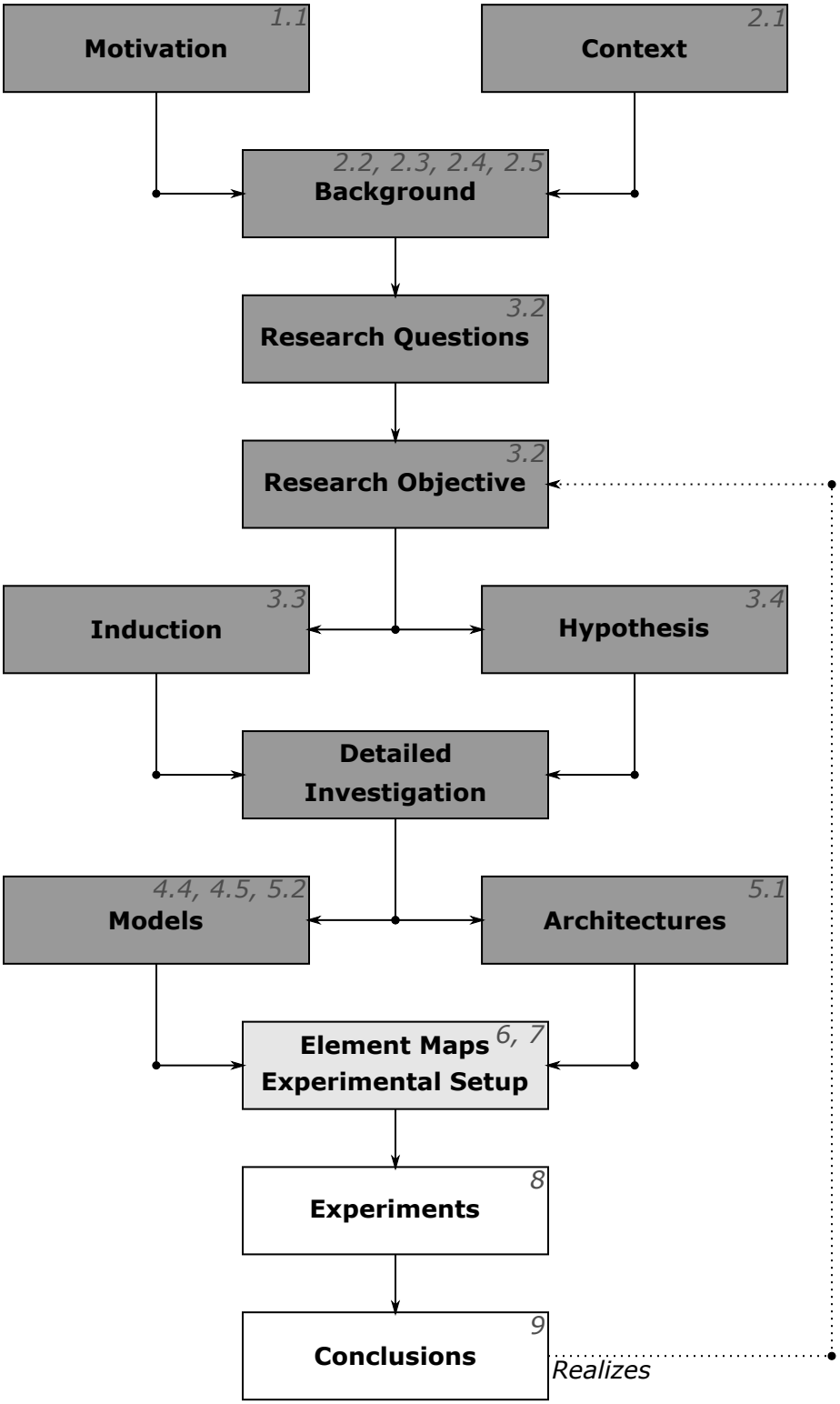


Figure 46: This chapter and next are used to go though each architecture view and determine their modeling potential.

Table 3: Model elements

Model Type	Elements
Graph	Vertex, Edge
Probability	Conditional Probability
System Dynamics	Stock, Flow, Variable
Markov Chain	States, Transitions
Petri Net	Places, Transitions, Arcs
Queueing	Arrival, Size, Server
Discrete Event	Event, Queue, Transition, Server, Entity, Resource
Agent-based	Agent, Environment, Interaction, Rules

It must be kept in mind that architecture views containing enough information to structure a computer model will be accepted here. The specific values that go into the actual simulation as inputs will not be required for acceptance. For example, an OV-5b includes many activities strung together to create a larger operational activity. Each of these activities have parameters such as how long it takes to finish each of them. However, such numeric information is usually not included in architecture views. In fact there are two specific architecture views that carry metric information: SV-7 and SvcV-7. Were all views ignored as *cannot be modeled* due to the lack of numerical information, there would be no information created in this study. Therefore, the requirements were relaxed slightly and systematically across the board to not require numerical values for modeling. However, the reader is to remember that model structures themselves cannot be simulated, and keep in mind that numerical values will be needed to perform any computer analysis eventually.

The rest of this chapter and the next will go though each architecture viewpoint type and determine their modeling potentials. This process is simply going through a list with no particular order and the logical development of the research is paused until the end of the lists are reached. Figure 46 shows the stage of the progress so far. Each map that is created between views and modeling types will have one letter designation: “Y”, “N”, or “M”. These represent yes, no, and maybe respectively. A

yes entry means that this architecture element can be translated into this modeling element. No means the opposite; maybe is inconclusive.

6.1 OV-1 High level operational concept graphic

The high level operational graphic shows the way the system of system works in a given scenario [59]. OV-1 usually shows the systems undertaking some actions in the context of a mission but also includes environment, organizations, facilities etc. in order to establish the context. For example, an OV-1 for close air support operations may show a limited area; however, an OV-1 for a deep interdiction mission will show a large geographical area. The OV-1 is generally ignored in modeling and simulation purposes because it cannot be used directly to model systems or their actions. However, it sets a context for the scenario; therefore, it is essential for modeling activities to start. OV-1 answers the question "what is being modeled". An example OV-1 is given in Figure 47.

There are examples in the literature that use the high level operational concept graphic as the starting point for modeling: AbuSharekh et al. use it to communicate what the model must simulate [12], Xiong et al. [108] and Wagenhals et al. use it in the first few steps of architecting and evaluation process [190]. Baumgarten and Silverman use OV-1 among many other operational views as static documents that are used to create discrete event simulations that are very detailed [28]. One must admit that there are also examples that do not use OV-1 explicitly. Kilicay-Ergin and Dagli do not mention a mission concept or scenario definition [116]. This is not surprising as their system of systems in question is not a directed (refer to page 21 for the definition). However, they do define a "high-level meta architecture" that shows the context of their models, which one can call an OV-1 easily. Another example of not using OV-1 is Rodrigues' work [170]. He gives an explanation of the SoS to be modeled in text and jumps right into the modeling components of it. The highest

level of architecture view used in the work is an SV-1 and serves as a general depiction of SoS operation.

Whether it is useful directly or indirectly for modeling efforts, it is highly likely that an architecture will include an OV-1. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a 92% creation rate for OV-1 among the projects considered [6]. It is the most commonly created product. According to Hurlburt, OV-1s are required documentation for all major DOD acquisition programs [103].

The elements that go on an OV-1 are not standardized by design to make the view flexible. Consequently, this architecture view cannot be decomposed into standard elements to be analyzed. However, it is safe to assume that it will include some *systems* as well as their *actions* that are part of the mission. It is also likely that some *facilities* are shown. An attempt was made to match these elements to elements found in modeling types. The following investigation is summarized in Table 204.

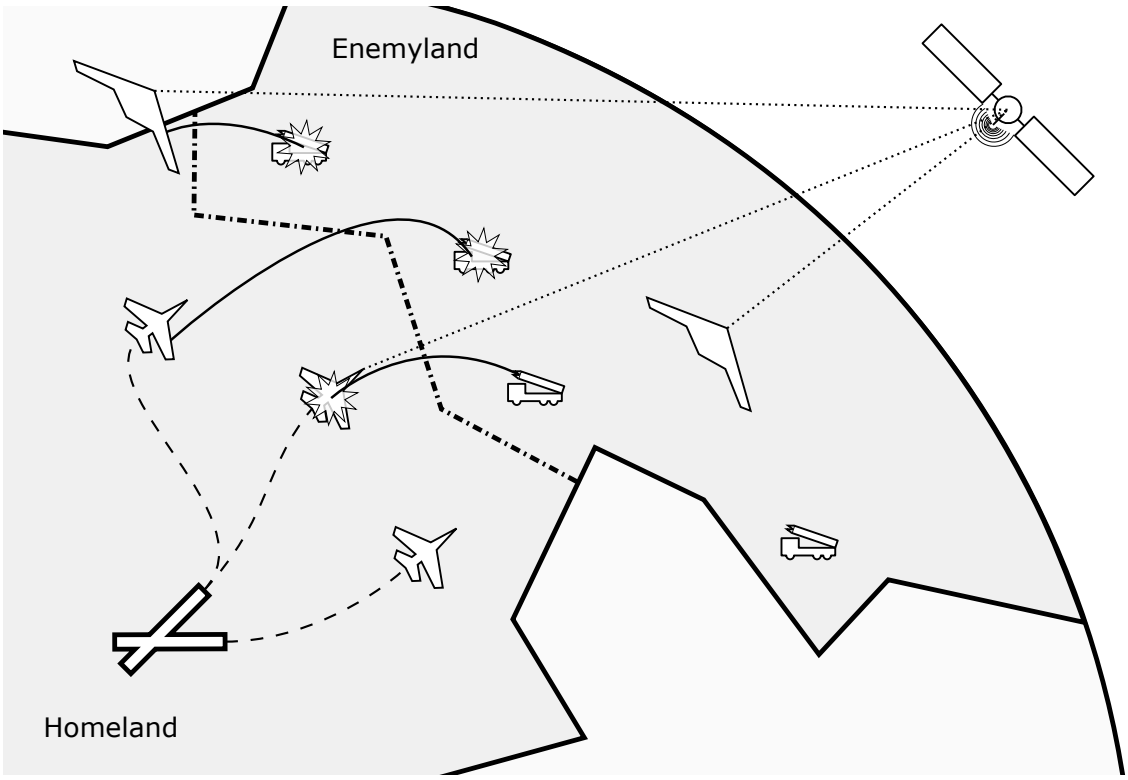


Figure 47: Example OV-1

6.1.1 Graph model

Graphs are very flexible models and most views will be modelable by graphs. Additionally, the choice between representing elements as vertices or edges in a graph model is determined by the number of connections the element has. An edge is always between two vertices by definition but a vertex may be connected to any number of edges. This rule determines what architecture element can be modeled as a vertex and what element as an edge. Usually, a connection for communication, resource exchange, or needline is between two operational nodes; and therefore, they are fit to be represented as graph edges. The reader must take note that under some conditions (e.g., a circular graph) architecture elements modeled as vertices can also be modeled as edges.

Because OV-1 usually depicts systems and their actions, it may be possible to represent actions as connections between the systems. In that simple case, it is plausible

that systems and facilities to be represented as vertices and actions or functions as edges that connect them to each other. The reader will find similar arrangements for OV-2, OV-5b, SV-1, SV-2, and SV-4. Additionally, because SV-5b is a more systematic way of displaying most of the information included in an OV-1, its graph model will be more structured than and OV-1's graph model. An example of how an OV-1 can be represented as a graph is shown in Figure 48. Table 4 sums up the findings.

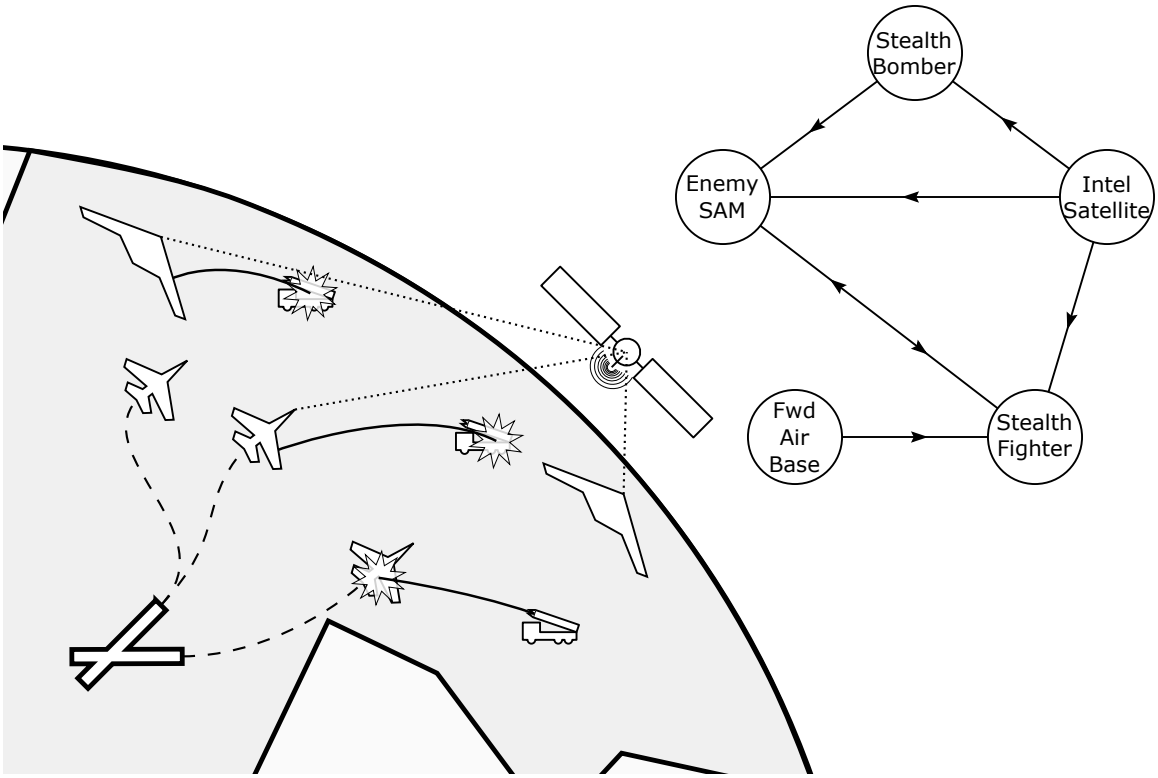


Figure 48: Example OV-1 to graph model translation

Table 4: Mapping between OV-1 and graph model elements

	Vertex	Edge
Systems	Y	N
Actions	N	Y
Facilities	Y	N

6.1.2 Probability model

A probability model from an OV-1 is technically possible; however, in many cases it would be indistinguishable from a graph model. OV-1s do not include numerical data but communicate a concept of operation; therefore, a probability model can be constructed but the necessary values would be missing on an OV-1. Also, the probability model may have missing elements that are included in the graph model because those nodes may not make sense to be modeled as a conditional probability. The system modeler should look into other architecture views to create a probability model. Figure 49 shows a possible OV-1 to probability network transformation. Similar to before, Table 5 sums up the findings.

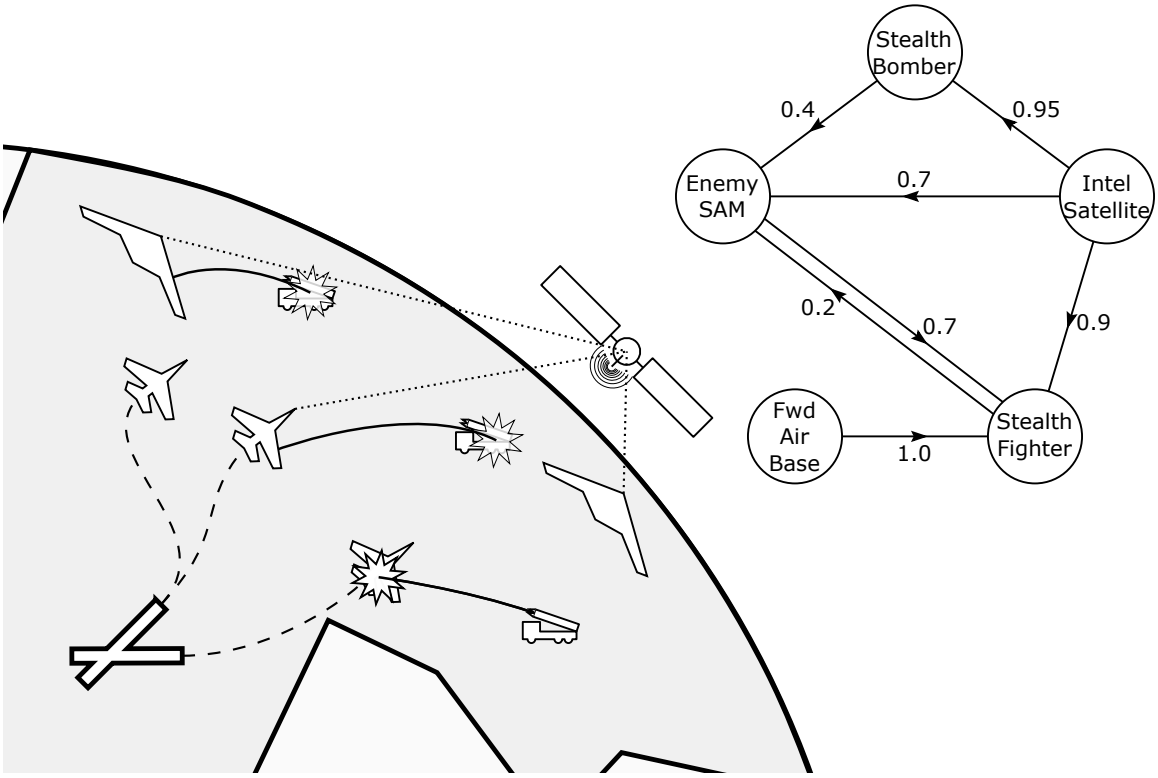


Figure 49: Example OV-1 to probability model translation

Table 5: Mapping between OV-1 and probability model elements

	Conditional Prob.
Systems	N
Actions	Y
Facilities	N

6.1.3 System dynamics model

System dynamics modeling and the following modeling types with OV-1 will have the same theme: there is not enough information to construct a model. However, the OV-1 is still a valuable architecture view for modeling purposes even if it does not include all the necessary detailed information as discussed before.

If one can imagine the large system of holding amounts of things in various stocks, and the exchange of things between the stores, a system dynamics model can be a natural way of representing the system. Based on this, systems and facilities can act as stocks for things that are exchanged and transported. For example, airports, en-route aircraft, parking lots, etc. can be modeled as stocks in a system dynamics formulation. In this example, passengers can be traded between the systems, hence they are the flows. Finally, the flows and stocks may have several variables affecting their performance. Figure 50 depicts another example of how airplanes can be treated as flows between flight information regions. All such elements do exist in an OV-1 but they are not detailed enough to build an entire system dynamics model. There appears to be no cohesive theme to interpret the graphical icons in an OV-1 as system dynamics modeling elements. The findings are summarized in Table 6.

Table 6: Mapping between OV-1 and system dynamics model elements

	Stock	Flow	Variable
Systems	Y	N	M
Actions	N	Y	N
Facilities	Y	N	M

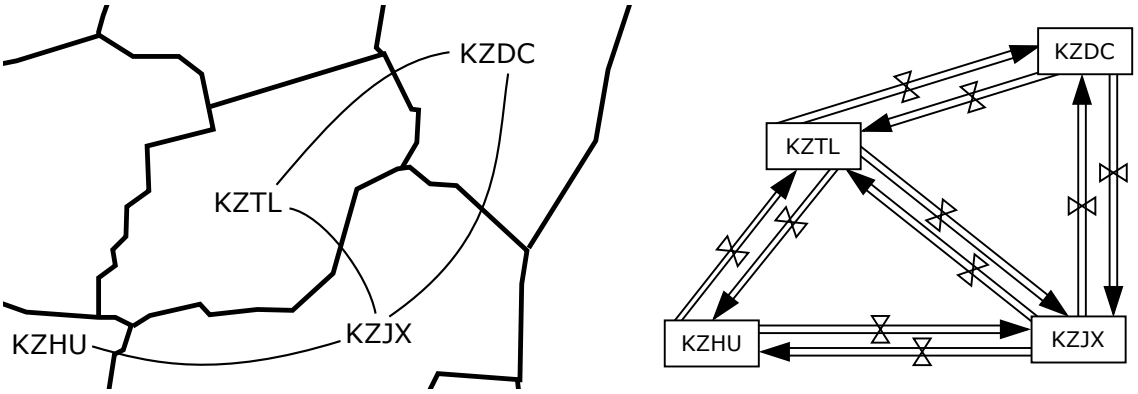


Figure 50: Example OV-1 to system dynamics model translation

6.1.4 Markov chain model

Markov chains represent possible states of a system and the transitions between the said states. An OV-1 depicts an operational concept and this includes system in certain states; however, the OV-1 does not usually include a description of how the systems transition between the states. In cases where such transitions are described, an OV-1 can be used to form a Markov chain.

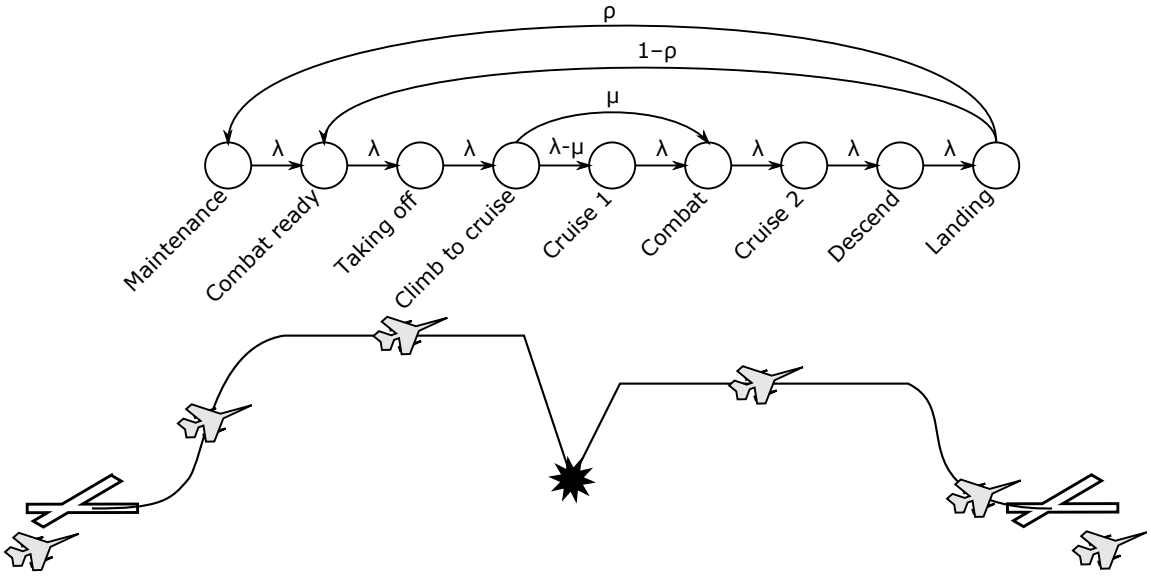


Figure 51: Example OV-1 to Markov chain model translation

A mission profile, which is a very common aerospace example, is given in Figure 51. The mission profile can be thought of as an OV-1. The numerical values for transitions

may be missing from the diagrams, but for the purposes of this analysis the specific values had been scoped out nevertheless. A structure for a Markov chain model can be constructed from OV-1 architecture views. The findings are summarized in Table 7.

Table 7: Mapping between OV-1 and Markov chain model elements

	State	Transition
Systems	N	N
Actions	N	Y
Facilities	Y	N

6.1.5 Petri net model

Petri nets are very good at modeling systems that change shape, form, mode, state, and merge with other systems or split into multiple systems. Many OV-1s can be interpreted using such statements. For example, an *attack aircraft carrying an air-to-ground missile* can split into an *attack aircraft not carrying a missile* and a *missile in flight*. Later a *missile in flight* and its *target* can merge into a *hit target*. Because such sentences are shown in an OV-1 graphically, there is enough information to be gained from it for the purpose of Petri net modeling. Systems and facilities can be thought of as places and actions can form transitions as well as arcs. These points are summarized in Table 8. An example transformation is given in Figure 52. AbuSharekh et al. uses an OV-1 to shape a Petri net model; however, most model information actually comes from other views in their work [12].

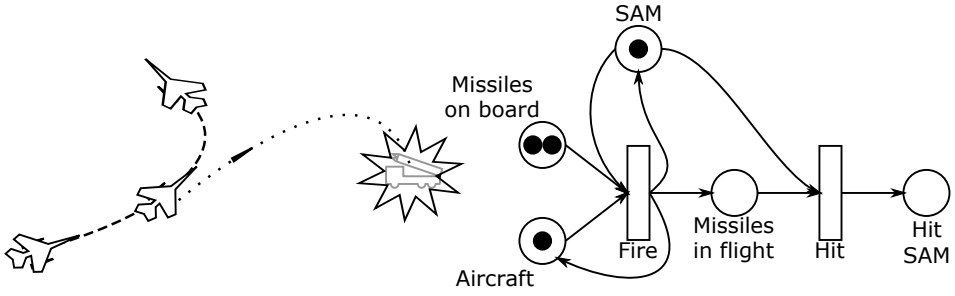


Figure 52: A part of an example OV-1 to Petri net model translation

Table 8: Mapping between OV-1 and Petri net model elements

	Place	Transition	Arc
Systems	Y	N	N
Actions	N	Y	Y
Facilities	Y	N	N

6.1.6 Queueing model

Queueing theory can be a quick analysis option for systems of systems with very specific properties that allow analytical solutions. The idea behind queueing models is that some jobs arrive at a workstation where servers perform actions to finish the jobs. An OV-1 depicts the concept of an operation; however, it lacks the details of each action within the operational concept. For example, it does not include much detail on how many of each job is required or can be generalized, it does not include concepts such as queueing which is the very focus of queueing models. Because queueing theory is dependent on specific assumptions to be practical to use compared with Markov chains or more complex discrete event models, using the OV-1 for queueing theory does not make much sense. The information included in an OV-1 can still give the modeler a rough idea as usual. The findings are summarized in Table 9.

Table 9: Mapping between OV-1 and queueing model elements

	Arrival	Size	Server
Systems	M	M	Y
Actions	Y	M	N
Facilities	N	N	Y

6.1.7 Discrete event model

Discrete event models are the pinnacle of operations research methods. They require a significant amount of knowledge about the system to set up. A depiction of the systems like the OV-1 will not be nearly enough to create a discrete event model; however, the OV-1 includes some relevant information for a discrete event model. Actions depicted in an OV-1 can be easily interpreted as events (also, queues and

transitions). The systems can in turn be servers or entities that either perform jobs or have jobs performed on them. And finally, facilities can also be represented as servers or resources, whichever is more suitable.

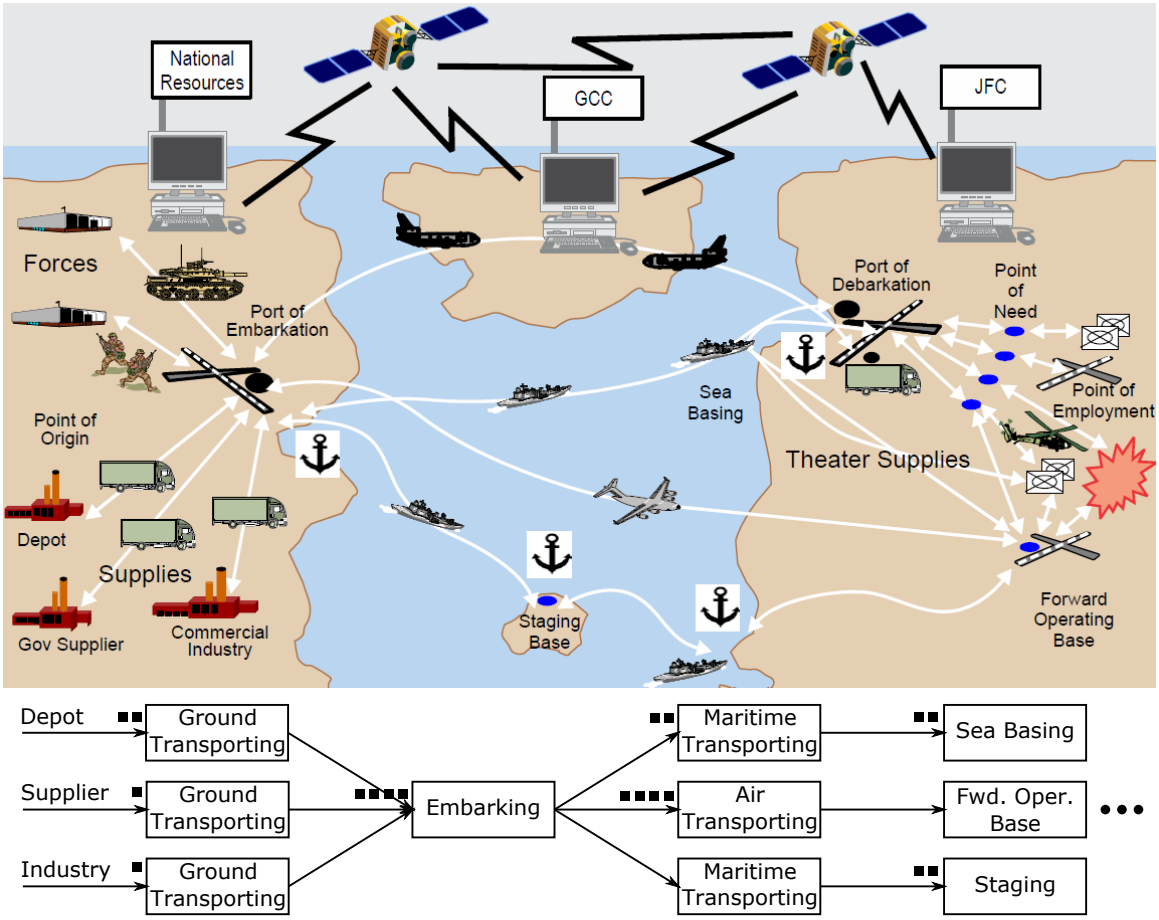


Figure 53: A part of an example OV-1 to discrete event model translation. The OV-1 was taken from Joint Publication 4-09 Distribution Operations[186].

While it is surprising to find discrete event models well catered to by OV-1s, the reader must accept that the detail needed to get an executable discrete event model will be mostly missing from any conceivable OV-1. The literature includes examples of OV-1s being used to create DES models [25, 28, 108, 141]; however, in all such cases, this particular view is only used to get the modeling activities started. In no example is an OV-1 used in isolation to create a discrete event model. Additionally, there are views (OV-6b and OV-5b) very well suited for discrete event modeling and

engineers should look into those solutions for the purpose of discrete event modeling rather than the OV-1. An example of how an OV-1 can provide context to discrete event modeling is given in Figure 53 and the findings are summarized in Tables 204–207 given in Appendix A.

Table 10: Mapping between OV-1 and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
Systems	N	N	N	M	Y	N
Actions	Y	M	N	N	N	N
Facilities	N	N	N	M	N	M

6.1.8 Agent-based model

Agent-based models have the highest complexity among the models discussed in this thesis. As previously stated in Chapter 5, they are most valuable if a significant amount of detail is known about the way individual agents behave in the system of systems rather than the behavior of the collective system of systems. Therefore, OV-1 may be the wrong kind of view for agent-based modeling. On the other hand, the view depicts the desired behavior for the final system of systems design. Keeping that and the fact that an OV-1 sets a context for the rest of the engineering effort in mind, the view can be used in an agent-based modeling effort. For example, systems depicted on OV-1s can be defined as agents, whose rules will be determined later. Actions in an OV-1 can form part of agent interactions with other agents or even the environment. However, they lack the ability to describe interaction rules in enough detail and cannot be used for rules. Facilities can be interpreted as agents or environment. An example view-to-model process is depicted in Figure 54 and the findings are summarized in Table 11.

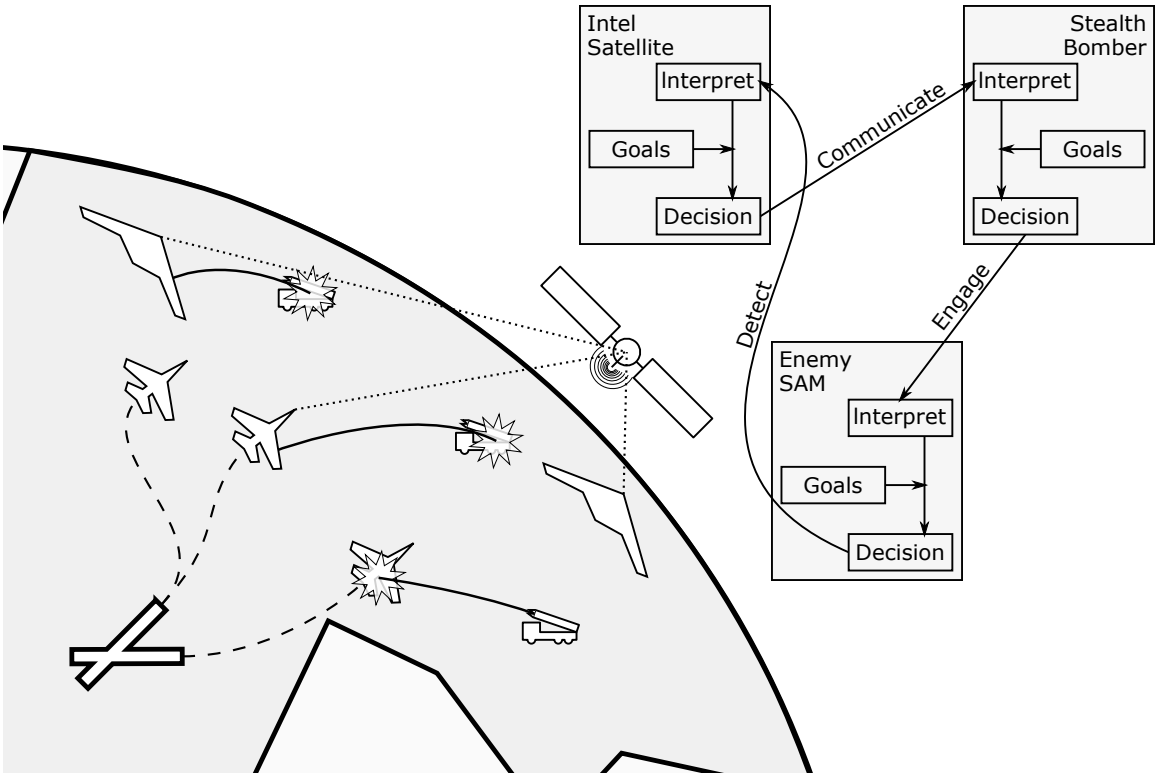


Figure 54: A part of an example OV-1 to agent-based model translation

Table 11: Mapping between OV-1 and agent-based model elements

	Agent	Environment	Interaction	Rules
Systems	M	N	N	N
Actions	N	N	M	N
Facilities	M	Y	N	N

6.2 OV-2 Operational resource flow description

OV-2 is an immensely useful view for planning that shows resource needs of operational nodes to function properly [59]. For example, a command and control node would require information resource flows from other operational nodes and those nodes would need orders communicated back. However, OV-2 is not a view particularly useful for modeling purposes because it skips details on how the resources are exchanged. DoDAF manual states that "it is to describe who or what, not how" [59]. This view can be considered as a requirement statement, not a detailed achievement

of the required communication links. One interesting use of an OV-2 can be the presentation of results from detailed analyses aggregated to OV-2's level of abstraction. It can then be used to check for deficiencies from the requirements. It has been used by Baumgarten and Silverman as a static description of the operations for modeling purposes [28]. Domerçant reads the OV-2 to calculate possible ways of rearranging operations using mathematical graph methods [64].

As defined in the DoDAF manual, the OV-2 has two elements: operational nodes and needlines. These elements are matched against modeling elements as was done with the OV-1. The results are summarized in Tables 204–207 given in Appendix A. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a very high creation rate (76%) for OV-2 among the projects analyzed [6]. According to Hurlburt, OV-2s are required documentation for all major DOD acquisition programs [103].

6.2.1 Graph model

A graph model is the best fitting modeling type for an OV-2. The view is a collection of *nodes* and *lines* and its translation to a graph model is immediate: nodes→vertices and lines→edges. When looked at graphically, their resemblance is striking (e.g., Figure 55 required a caption to distinguish the view from the models). The resulting graph model will not be representative enough to make engineering decisions such as improving efficiency or effectiveness of the system of systems, because OV-2 does not include implementation details as discussed earlier. However, requirement compliance can be checked using a graph model. The results of the model should not be taken as entirely positive without considering the lack of implementation details in the OV-2s. The findings are summarized in Table 12 and Figure 55 shows an example OV-2 to graph model transformation. The example OV-2 is based on the As-Is version of the National Airspace System Enterprise Architecture[149]. Because the original OV-2

includes two types of needlines, two graph models are created from it.

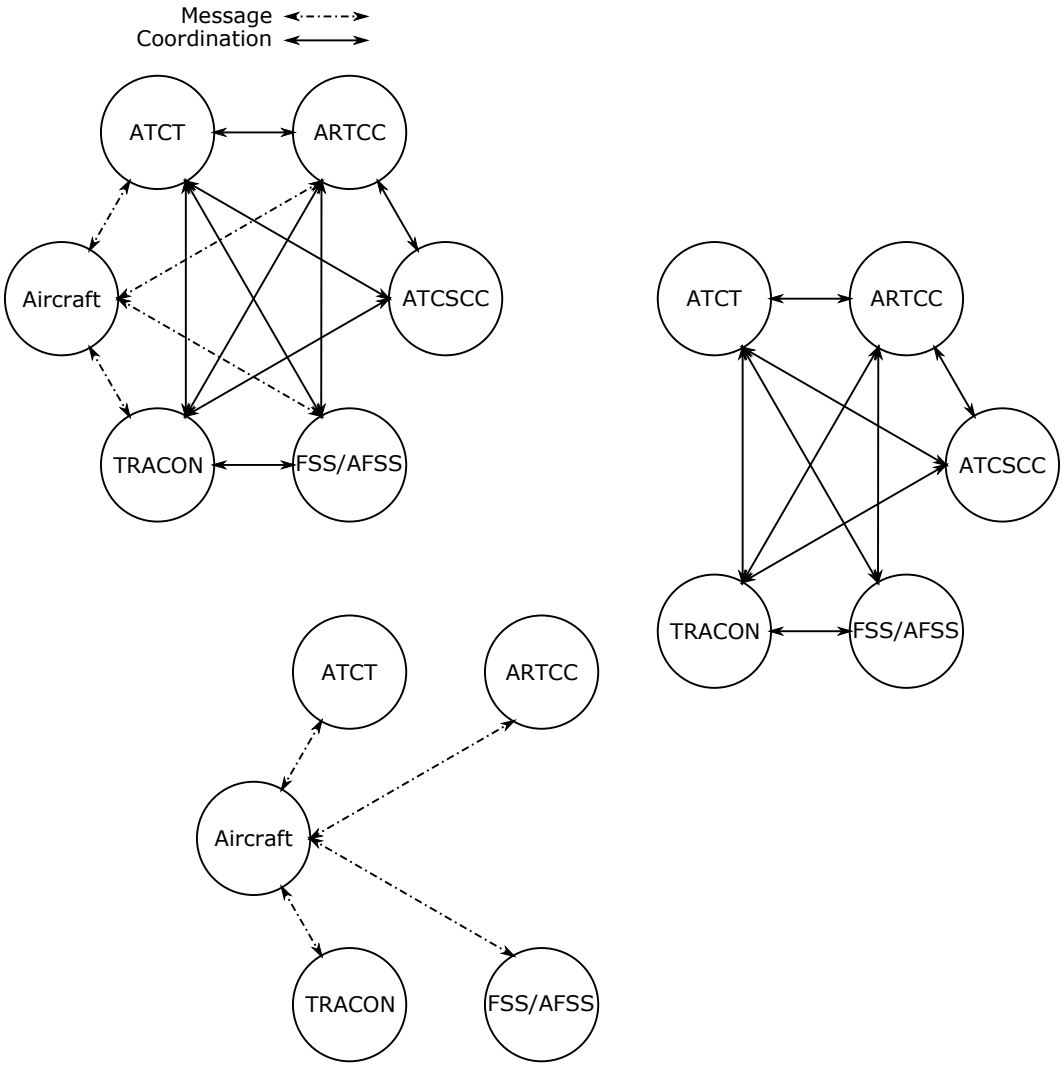


Figure 55: A part of an example OV-2 to graph model translation. The original OV-2 is on the top left.

Table 12: Mapping between OV-2 and graph model elements

	Vertex	Edge
Operational node	Y	N
Needline	N	Y

6.2.2 Probability model

All models except the graph model should be used very carefully with the OV-2. Probability of messages, resources etc. reaching their intended operational sink nodes depends on the way they are transmitted. However, the OV-2 does not carry such information. OV-2 only shows the nodes in need of transmitted things and the sources of the transmitted things. *How* the transmission occurs is abstracted away entirely. Table 13 summarizes the findings and Figure 56 shows an example. An OV-2 derived probability model would make a very suitable requirements compliance method. If the operational nodes are required to have a certain probability to be connected, after the higher-fidelity analyses are performed, the resulting probabilities can be shown on the OV-2 allowing direct comparison for the decision maker.

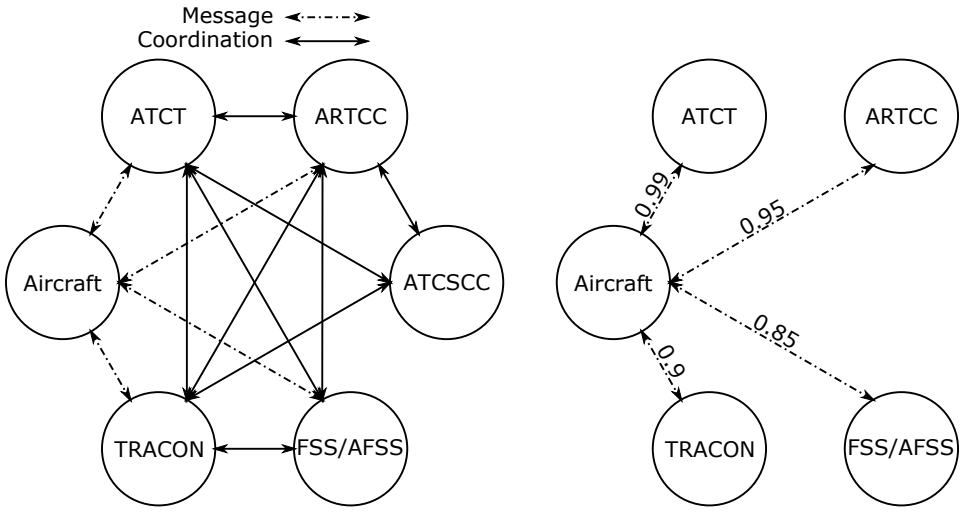


Figure 56: A part of an example OV-2 to probability model translation

Table 13: Mapping between OV-2 and probability model elements

	Conditional probability
Operational node	M
Needline	M

6.2.3 System dynamics model

System dynamics models are another high-level model that bear some resemblance to OV-2 views. However, much like the probability models, their potential use is limited in modeling the system of systems because of the lack of crucial details in an OV-2. The dynamics learned from an OV-2 would be entirely misleading because the needlines do not actually represent information or material transfer but the need of them in operational activities. However, it is not impossible to use OV-2s for system dynamics modeling. In some simpler cases, where the actual flow of information and material resembles the operational needlines, it might be possible to turn an OV-2 into a meaningful system dynamics model. Table 14 summarizes the findings.

Table 14: Mapping between OV-2 and system dynamics model elements

	Stock	Flow	Variable
Operational node	M	N	N
Needline	N	M	N

6.2.4 Markov chain model

An OV-2 can be translated into a Markov chain form easily due to their graphical resemblance but a Markov chain model from an OV-2 would also be too simplistic to represent much for the system of system it describes. Because of the same reasons discussed earlier, the model would be unable to capture any meaningful dynamical evolution of the system. Additionally, the OV-2 does not necessarily depict a state changing system, i.e., all operations depicted on the view are most likely happening concurrently. The system engineer must be careful about what he/she is modeling with a Markov chain. For example, a single piece of information may be tracked

through the operation to understand when and where it is being used; however, this would only model a tiny part of the whole system of systems. Table 15 summarizes the findings.

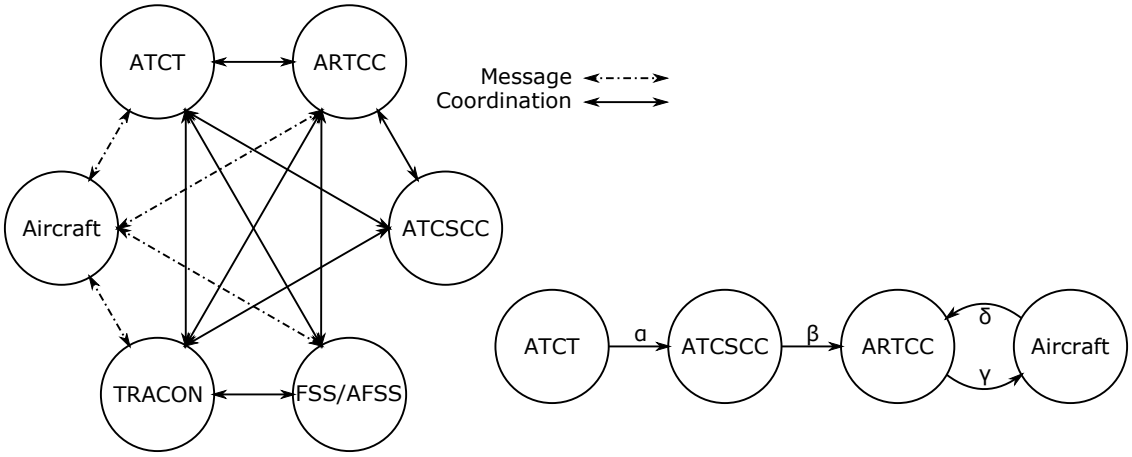


Figure 57: A part of an example OV-2 to Markov chain model translation

Figure 57 shows an example of how adverse weather information can create a series of events leading to incoming aircraft diverting to another airport. The destination airport's tower coordinates with the air traffic control system command to divert all incoming aircraft. The command center then works with the regional air route traffic control centers. The aircraft is notified by the control center via a radio message that it needs to change its trajectory. The aircraft complies with the instructions.

Table 15: Mapping between OV-2 and Markov chain model elements

	State	Transition
Operational node	M	N
Needline	N	M

6.2.5 Petri net model

The OV-2 does not include any information on how the resources gathered from the needlines end up being used, whether they are merged and passed on to other nodes etc. Therefore, a Petri net model of an OV-2 would not be highly useful. Table 16

summarizes the findings. If a Petri net is going to be created because other views that are suitable for Petri nets exist, an OV-2 can provide some information on how to structure the model. The OV-2 and the Petri net formalism look fairly similar. As can be seen in Table 16, the arcs and transitions are defined from a single architecture element, which is not ideal.

Table 16: Mapping between OV-2 and Petri net model elements

	Place	Transition	Arc
Operational node	M	N	N
Needlines	N	M	M

6.2.6 Queueing model

Queueing models can be a good fit to represent the ideas behind what OV-2 views are depicting; however, the lack of implementation details renders them useless in the same way the Petri nets or Markov chains are useless. Because a queueing model requires information on the kinds of jobs performed on arriving work packages, OV-2s are not practical for Petri net modeling. Table 17 summarizes the findings that are entirely negative.

Table 17: Mapping between OV-2 and queueing model elements

	Arrival	Size	Server
Operational node	N	N	N
Needlines	N	N	N

6.2.7 Discrete event model

Discrete event models are further complicated compared with queueing models. The OV-2 abstracts away many interesting nuances of the system that can be investigated by a discrete event formulation. Therefore, the information included in an OV-2 is mostly vaguely guiding and not specific enough for discrete event modeling. The information in an OV-2 that could be useful will be repeated again in a more suitable

view such as the OV-5b as well. It is best if OV-2 is not considered for discrete event modeling purposes. Table 18 summarizes the findings.

Table 18: Mapping between OV-2 and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
Operational node	M	M	N	M	N	N
Needlines	N	N	M	N	M	N

6.2.8 Agent-based model

An OV-2 has a very high level perspective of the system of systems. This is a major mismatch for agent-based modeling, which requires detailed, low-level information on how its agents must behave and is used to discover emergent larger-scope behavior. Additionally, the lack of implementation details in an OV-2 renders the view entirely impractical for an agent-based modeling effort. OV-2's system view counterpart SV-1 will have some—still limited but some—use but an OV-2 is better left alone for agent-based modeling purposes. Table 19 summarizes the results.

Table 19: Mapping between OV-2 and agent-based model elements

	Agent	Environment	Interaction	Rules
Operational node	M	N	N	M
Needline	N	N	N	M

6.3 OV-3 Operational resource flow matrix

OV-3 is a matrix representation of the basic information included in an OV-2 with the addition of more details of why the resource flows are needed. The findings from an OV-2 apply to the OV-3 for the discussion of its use in modeling almost exactly. The only difference is that some additional detail on the exchanges is included in an OV-3 which is its purpose. It is presented in a table form listing each exchange shown in the OV-2 row-by-row. In each row of an OV-3, there is information on the origin operational node and destination operational node and details on the resource

exchanged. The details could be significantly different based on the system of systems in question. It is difficult to pass judgment on whether the details would be useful for various modeling types. It is promptly scoped out of this discussion. Baumgarten and Silverman follow the OV-1 to OV-3 to OV-5 approach when weaving detail into their executable architectures [28].

The source and sink operational nodes are the same kinds of things so they are represented with on simple operational node in the analysis. Additional details about the resource exchange can be added as appropriate. The details on how the exchange is implemented are non-existent just like the OV-2. Because the similarities are striking, most of the discussion that follows is abridged to minimize repetition. The results are summarized in Tables 204–207 given in Appendix A. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a high creation rate (66%) for OV-3 among the projects analyzed [6]. According to Hurlburt, OV-3s are required documentation for all major DOD acquisition programs [103].

6.3.1 Graph model

The mapping of OV-3 elements to graph elements is essentially the same process as the mapping of OV-2 to graphs. The details included in an OV-3 table could potentially be used to assign numbers to graph edges to create weighted graphs, but the basic premise is the same. Table 20 shows the results. Figure 58 depicts a conceptual example.

Table 20: Mapping between OV-3 and graph model elements

	Vertex	Edge
Activity	Y	N
Exchange	N	Y

Exchange ID	Information Element Name	Sending Node	Receiving Node
TOWER-ATCSCC-I/O-2	Wx Status	ATCT	ATCSCC
ATCSCC-ARTCC-I/O-1	Traffic Flow Plan	ATCSCC	ARTCC
ARTCC-AIRCRAFT-I/O-2	Advisory	ARTCC	AIRCRAFT

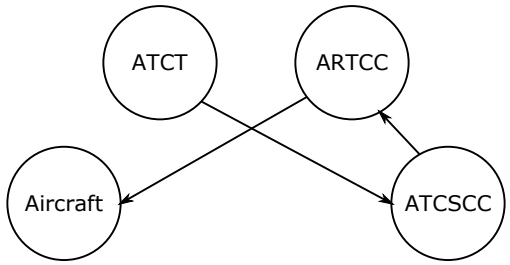


Figure 58: A part of an example OV-3 table to graph model translation.

6.3.2 Probability model

If the details on the resource exchange included are probabilities, the probability model could be filled in with more information compared with a probability model made from an OV-2. Otherwise, the results are the same. Table 21 summarizes the results.

Table 21: Mapping between OV-3 and probability model elements

	Conditional probability
Activity	M
Exchange	M

6.3.3 System dynamics model

Each row in the OV-3 can be modeled as a flow in the system dynamics formalism. However, as in the case of OV-2, the implementation details are lacking. The details included in the OV-3 rows are not enough to create a meaningful system dynamics model. Table 22 shows the results.

Table 22: Mapping between OV-3 and system dynamics model elements

	Stock	Flow	Variable
Activity	M	N	N
Exchange	N	M	N

6.3.4 Markov chain model

The extra details included in each row can be used to populate a more complete transition matrix. Additionally, a single OV-2 or OV-3 can actually create Markov chain models for each resource type, but at this high level of operations Markov chains are not well suited to be used as discussed under the OV-2 section. Table 23 summarizes the results.

Table 23: Mapping between OV-3 and Markov chain model elements

	State	Transition
Activity	M	N
Exchange	N	M

6.3.5 Petri net model

As discussed earlier, OV-2s are not highly suited to be used for Petri net modeling. The extra information found in the OV-3 table rows could offer hints in how resources are used. This can potentially fix the issues from the OV-2 analysis. However, without the specific information how the resources are combined, split, and used, OV-3s are still not well suited for Petri nets. Table 24 summarizes the findings.

Table 24: Mapping between OV-3 and Petri net model elements

	Place	Transition	Arc
Activity	M	N	N
Exchange	N	M	M

6.3.6 Queueing model

The extra information included in an OV-3 can help fill in some values in a queueing model. Otherwise, the results summarized in Table 25 are identical fo the OV-2 results.

Table 25: Mapping between OV-3 and queueing model elements

	Arrival	Size	Server
Activity	N	N	M
Exchange	M	M	N

6.3.7 Discrete event model

Because the OV-3 includes same details on the resources exchanged between operational nodes and how the resources are used, a hypothetical discrete event model may in an unlikely case explicitly define event timings and queueing times. Also, the exchanged information or resource may be used to define resources in the discrete event formulation. Baumgarten and Silverman use the OV-3 as a part of operational views package to create discrete event simulations[28]. In their work, it is used as a stepping stone from OV-1 to OV-5. Most of the information required to build their model comes from the pair of OV-5 and SV-2. Otherwise the analysis is identical to the OV-2 analysis for discrete event modeling. Table 26 shows the results.

Table 26: Mapping between OV-3 and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
Activity	M	M	N	M	N	N
Exchange	M	M	M	N	M	M

6.3.8 Agent-based model

The same mismatch between agent-based formulation and the high-level OV-2 repeats itself for the OV-3. Apart from the potential details about the interaction through the resource exchange no new substantial information is gained and OV-3 is deemed to be not particularly useful for agent-based modeling purposes still. Table 27 summarizes the results.

Table 27: Mapping between OV-3 and agent-based model elements

	Agent	Environment	Interaction	Rules
Activity	M	N	M	M
Exchange	N	N	M	M

6.4 OV-4 Organizational relationships chart

OV-4 shows an organization and its constituents. It is sometimes known as an organization chart or org chart. The operation depicted in an OV-1 is usually performed by heterogeneous systems and services belonging to different organizations, divisions, arms, etc. (i.e., sub-organizations). Figure 59 shows an example OV-4. The shaded region will be used for modeling examples below. For example, a distribution company would be connected to suppliers and clients externally and be organized as sales, customer support, warehouse, accounting, human resources, management teams internally. The resulting inevitable tree structure can be used to analyze the organization's agility for example. If sub-organizations are deep/vertical and are not connected with direct relationships, a job will need to go through many offices to get to the office tasked to perform it. An OV-4 is useful in such structural studies but includes almost no information about the details of how actions are performed.

There are two types of OV-4s. The first shows actual departments, actual sub-organizations whereas the second type only shows typical roles and posts. For systems of systems where actions follow an organizational order, OV-4s are extremely important. Mathieu and Callaway use an OV-4 diagram to guide their modeling of power distribution [134]. The results are summarized in Tables 204–207 given in Appendix A. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a high creation rate (68%) for OV-4 among the projects analyzed [6].

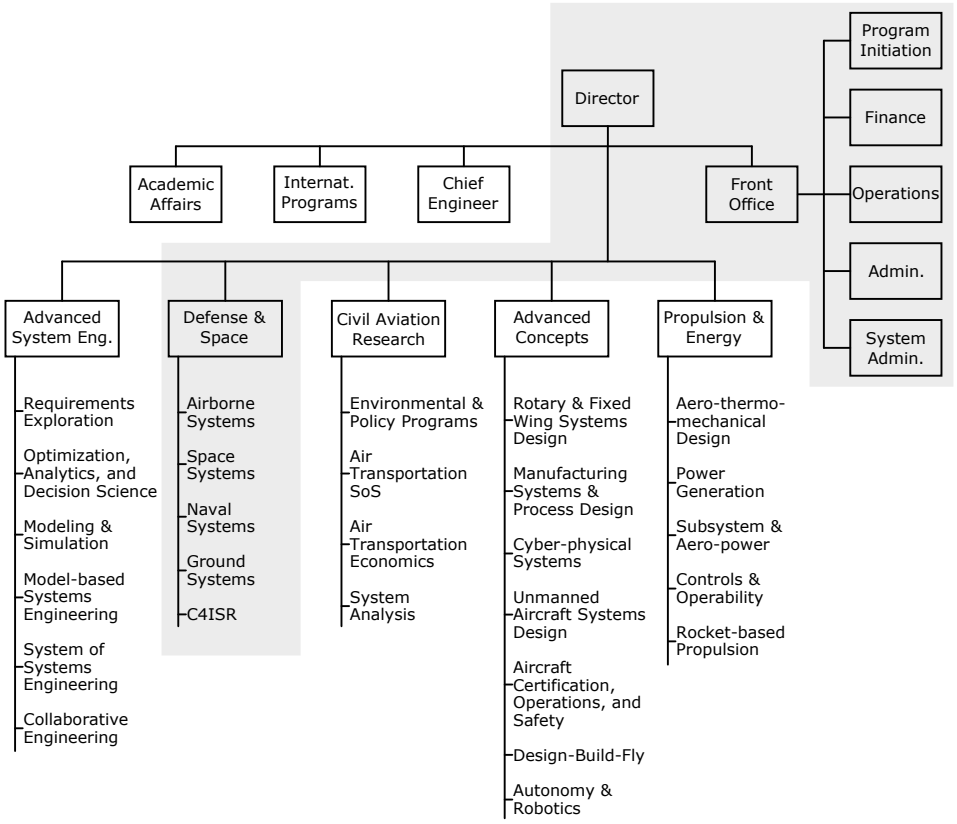


Figure 59: Example OV-4. Shaded region will be used in the examples.

6.4.1 Graph model

A graph model is perfectly suited to perform analyses using an OV-4. It can be used to measure organizational distances between sub-organizations. This distance should be minimized for roles that work on the same type of jobs: cooperation can be enabled through proximity. It can also be used to estimate consequences of reorganization efforts (creating more relations or breaking them). In general, it can also measure geodesic distances, eigenvalues, and sizes for the analysis of various dynamical properties of the organization. It is natural to use vertices for organizations and edges for relations in the OV-4 context. Domerçant uses OV-4s to differentiate systems of systems with the same workflow or systems but different hierarchies [64]. The results are summarized in Table 28. A hypothetical example is given in Figure 60.

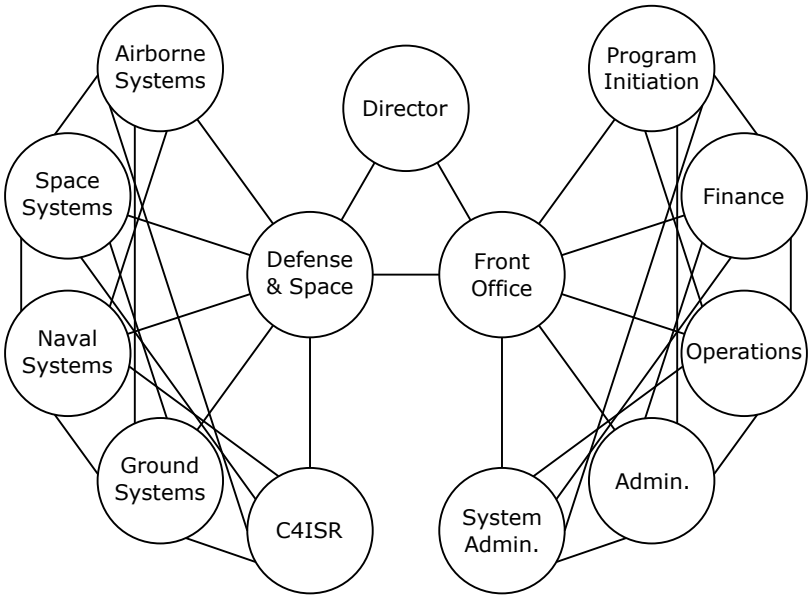


Figure 60: Example OV-4 to graph model translation

Table 28: Mapping between OV-4 and graph model elements

	Vertex	Edge
Organization	Y	N
Relationship	N	Y

6.4.2 Probability model

The relations between organizational nodes are not related to probabilities; therefore, constructing a probability model from the information given in an OV-4 is not possible. Engineers wanting to create probability models should look into other architecture views. Table 29 summarizes the results.

Table 29: Mapping between OV-4 and probability model elements

	Conditional probability
Organization	N
Relationship	N

6.4.3 System dynamics model

OV-4 may not be the best way to describe how work transitions from one actor to another for specific operations; however, it is a good way to measure an organization's

effectiveness as a whole. In order to measure this effectiveness, the jobs performed and being transitioned to others could be analyzed using an OV-4 diagram and system dynamics. Loops that are reinforcing within the organization may be discoverable and eliminated. In the system dynamics context, reinforcing loops create extra jobs internally as they are cycled through the system, not finish them. Therefore, they are not desirable in the OV-4 context.

An OV-5 would be a much better fit for specific operations and should be used before an OV-4 to set up a system dynamics model. The organizational nodes could be represented as stocks where jobs pile up and the relations between organizational nodes represent the transition of jobs to other nodes. Table 30 summarizes the results.

Table 30: Mapping between OV-4 and system dynamics model elements

	Stock	Flow	Variable
Organization	M	N	N
Relationship	N	M	M

6.4.4 Markov chain model

Similar to the discussion on OV-4 used for system dynamics. Markov chains can be used for individual jobs. Each job can be tracked probabilistically through the organization. Using a Markov analysis the time ranges (e.g., earliest, latest) to finish a job with a set confidence can be found. However, much like the system dynamics, an OV-5b would be a better view to use for the purpose of building a Markov chain model. One exception to this point is systems of systems whose operations follow the organizational structure very closely. Mathieu and Callaway model a power distribution network using Markov chains with the information included in an OV-4[134]. Based on this discussion, the OV-4 to Markov chain mappings are dotted with *maybe* entries. Table 31 summarizes the finding.

Figure 61 shows the example of how a research project is initiated at the author's laboratory. An external project sponsor submits a request for proposal through a

channel. The director brings the proposal to program initiation for processing. Program initiation determines that the defense and space division is best suited for the project. Within the division, the naval systems branch is tasked to write a proposal. Based on the work proposed, a budget is prepared in the finance office. Program initiation puts the finishing touches and with the final approval of the director, the proposal is sent to the school's office of sponsored programs, which sends it to the project sponsor. If the sponsor accepts the proposal, it is sent back to the school's office of sponsored programs. They notify the program initiation and finance offices, and the program initiation notifies the naval systems branch to start the research. The reader will notice the step-by-step process, which is more suitable for OV-5bs. The same example will be explored in Section 6.6.4 for OV-5b to Markov chain translation.

Table 31: Mapping between OV-4 and Markov chain model elements

	State	Transition
Organization	M	M
Relationship	M	M

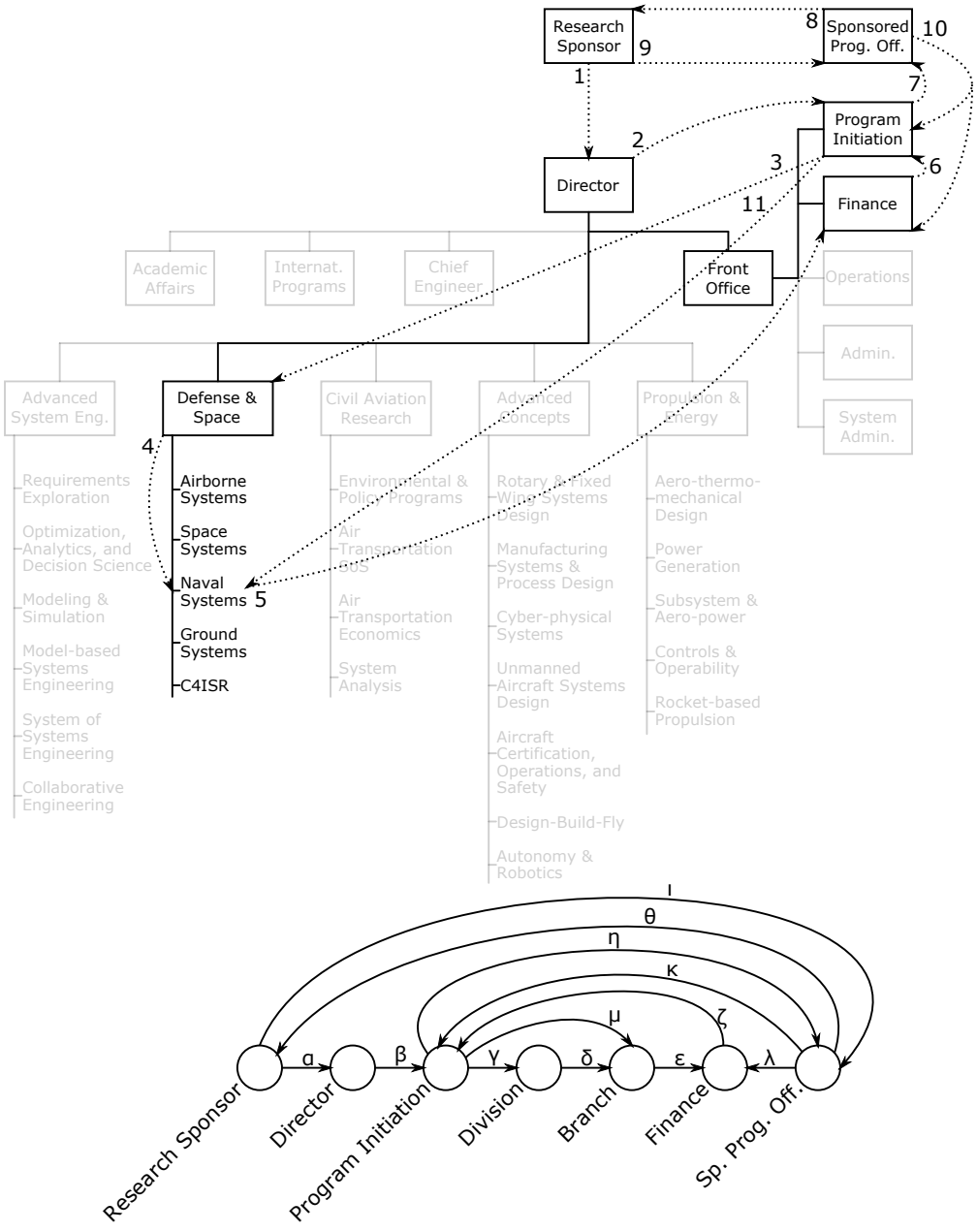


Figure 61: Example OV-4 to Markov chain model translation

6.4.5 Petri net model

A Petri net formulation for an organization chart makes little to no sense. The OV-4 shows separates between organizational entities whereas a Petri net would analyze the way jobs would merge or split. Table 32 summarizes the purely negative findings.

Table 32: Mapping between OV-4 and Petri net model elements

	Place	Transition	Arc
Organization	N	N	N
Relationship	N	N	N

6.4.6 Queueing model

Both queueing and discrete event models would seek detailing the organization's response to mode work arriving to it. However, the relations between organizational nodes are commonly hierarchical not sequential. It would be possible to use several OV-5 and OV-6s to perform queueing and discrete event analyses and display workload distributions on an OV-4. In that way, it could be a good visual analytics tool. Table 33 summarizes the findings.

Table 33: Mapping between OV-4 and queueing model elements

	Arrival	Size	Server
Organization	N	N	N
Relationship	N	N	N

6.4.7 Discrete event model

Discrete event models using OV-4s were discussed in the discussion for the queueing models in the previous section. Table 34 summarizes the findings.

Table 34: Mapping between OV-4 and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
Organization	N	N	N	N	N	N
Relationship	N	N	N	N	N	N

6.4.8 Agent-based model

An OV-4 is not a complete picture for an agent-based modeling effort. However, the hierarchy includes hints on agent types, agent interactions, and even possibly rules on their interactions. OV-4 would be a good view to consider when modeling a large organized operation along with views that detail the actual operation. The type of OV-4 that shows roles and posts can be useful in *subtyping* in object-oriented programming languages that are extremely suitable for agent-based modeling. Table 35 summarizes the results. Figure 62 shows how the elements map between the view and an agent-based formulation.

Table 35: Mapping between OV-4 and agent-based model elements

	Agent	Environment	Interaction	Rules
Organization	Y	N	N	N
Relationship	N	N	Y	Y

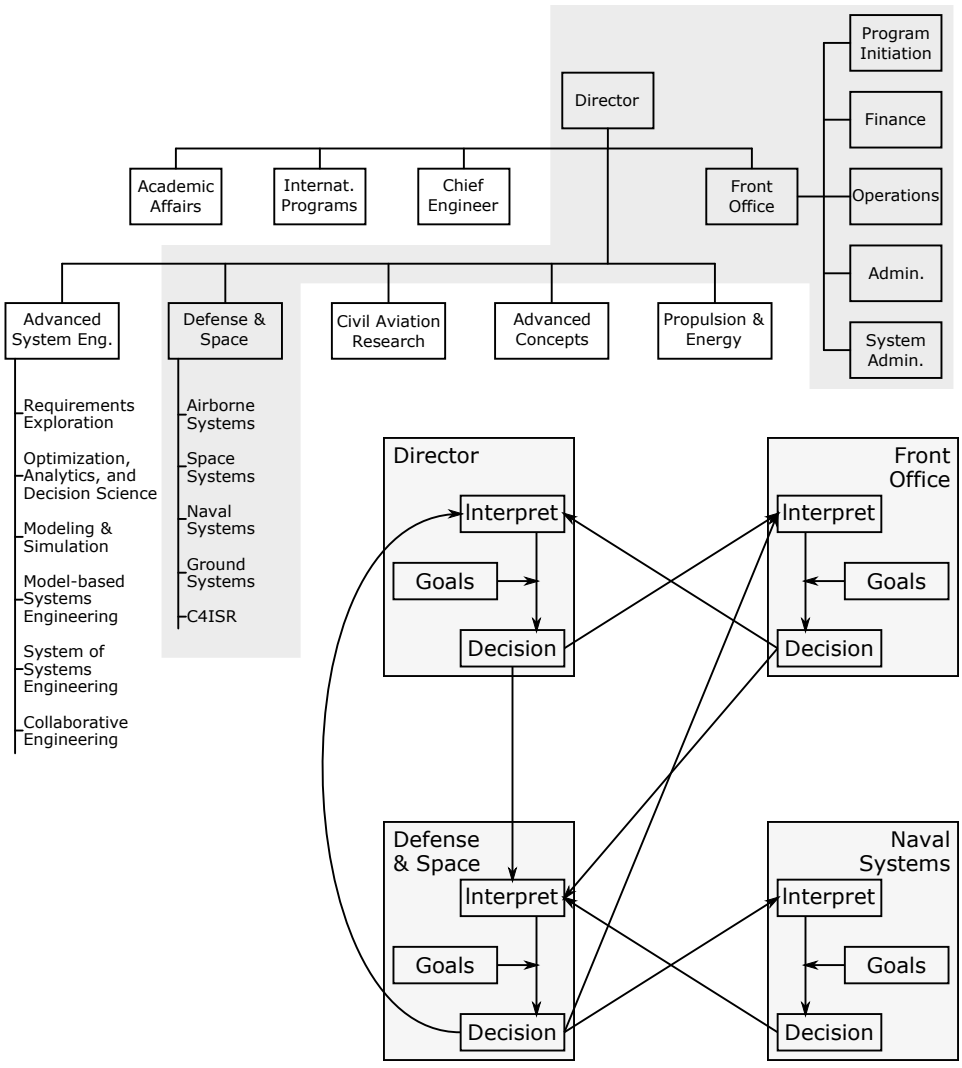


Figure 62: Example OV-4 to agent-based model translation

6.5 OV-5a Operational activity decomposition tree

OV-5a resembles the OV-4. It decomposes the operation being modeled instead of the organization performing the operation. An OV-5a is a further decomposition of the nodes shown on an OV-2 sans needlines. OV-5a is a good list for actions to be modeled; and therefore, it is a good checklist of preparation for a modeling effort. However, it does not depict the way these actions are taken; as such it is not very useful for modeling purposes. The literature is slightly difficult to interpret when it comes to the OV-5a or OV-5b because some authors simply refer to their architecture

views as OV-5. The language or included figures in some papers could be used to infer the version used but not always. The OV-5a is the rarer of the two and is less useful in structuring a model, although it can be useful in object oriented coding purposes. The results are summarized in Tables 204–207 given in Appendix A.

DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a high creation rate (63%) for OV-5a among the projects analyzed [6]. AbuSharekh et al. use a functional decomposition in their modeling efforts that is effectively an OV-5a [12], Mittal et al. use the OV-5a to list functions that together form capabilities and use the OV-6b for sequencing information [141], and Domerçant uses an OV-5a to group various operational activities [64].

6.5.1 Graph model

OV-5a works well with graph models for the same reason OV-4 works well with graph models. However, the relationship between operational nodes and their hierarchically lower or higher nodes do not have any practical use within graph models. The results are summarized in Table 36.

Table 36: Mapping between OV-5a and graph model elements

	Vertex	Edge
Activity	M	N
Relationship	N	M

6.5.2 Probability model

The relations between operational nodes on an OV-5a are not related to probabilities. Therefore probability models are not relevant after the creation of an OV-5a. The results are summarized in Table 37.

Table 37: Mapping between OV-5a and probability model elements

	Conditional probability
Activity	N
Relationship	N

6.5.3 System dynamics model

The hierarchical relations between operational nodes cannot be represented using a system dynamics model. The lines are not flows of quantifiable stuff. The analysis therefore returns entirely negative results and they are summarized in Table 38.

Table 38: Mapping between OV-5a and system dynamics model elements

	Stock	Flow	Variable
Activity	N	N	N
Relationship	N	N	N

6.5.4 Markov chain model

The system of system's operational state will not jump between operational nodes based on their hierarchy. Similar to system dynamics models Markov chain modeling efforts will not find OV-5as practically useful. Table 39 summarizes the results.

Table 39: Mapping between OV-5a and Markov chain model elements

	State	Transition
Activity	N	N
Relationship	N	N

6.5.5 Petri net model

The use of OV-5a for Petri nets is not practical for the same reasons that OV-4 was not practical. The reader is referred to Section 6.4.5 for details. Table 40 summarizes the results.

Table 40: Mapping between OV-5a and Petri net model elements

	Place	Transition	Arc
Activity	N	N	N
Relationship	N	N	N

6.5.6 Queueing model

The relationship lines between operation nodes present mainly hierarchical relationships. However, queueing models require sequential relations in order to model dynamics of system behavior. There is a major mismatch. The negative results are summarized in Table 41.

Table 41: Mapping between OV-5a and queueing model elements

	Arrival	Size	Server
Activity	N	N	N
Relationship	N	N	N

6.5.7 Discrete event model

Discrete event models are also not a good match for OV-5a views for the same reason that queueing models are not. Table 42 shows the negative results. Mittal et al. describe the use of an OV-5a in the process of creating a discrete event model but also note that the real source of information in the operational modeling comes from the OV-6 or other fit for purpose views[141].

Table 42: Mapping between OV-5a and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
Activity	N	N	N	N	N	N
Relationship	N	N	N	N	N	N

6.5.8 Agent-based model

OV-5a has no details on systems (agents) or the environment or interactions between them; therefore, it is not a good match for agent-based modeling efforts. One exception exists that must be discussed. The hierarchical organization of operational

nodes/activities can be potentially useful for object-oriented coding for agent-based rules. Object-oriented approach has very practical advantages for agent-based or discrete event modeling types. The view can be used for subtyping and inheritance purposes. Apart from this convenience, OV-5a is mostly useless for agent-based modeling efforts. Table 43 summarizes the results.

Table 43: Mapping between OV-5a and agent-based model elements

	Agent	Environment	Interaction	Rules
Activity	N	N	N	M
Relationship	N	N	N	N

6.6 OV-5b Operational activity model

OV-5b is an immensely useful architectural view for modeling purposes. It displays actions taken during operations and their order, dependencies, and events that trigger them. The view itself consists of activity, input and output, and some other miscellaneous architectural elements. These miscellaneous elements are defined vaguely out of necessity and can be thought of as notes or relevant details about the standard activity and input/output elements. OV-5b usually also includes systems that perform the activities, which ties it closely with the SV-4. DoDAF does not enforce a standard presentation but similar standards have been created before DoDAF existed and there are convenient and vetted methods such as IDEF0 that can be used for this view. An example OV-5b is given in Figure 63.

The activity elements are depicted as shapes that usually hold some important detail about what the activity is within the boundaries of the shape. Domerçant uses an OV-5b to organize a military mission into parallel and serial chunks [64]. Ultimately, the activities are actions that make up a part of the functions of the system of systems being described. These functions take in the information shown as arrows on the OV-5b and after processing it output another kind of information. The subsequent functions perform in the same manner until the end operational goal of

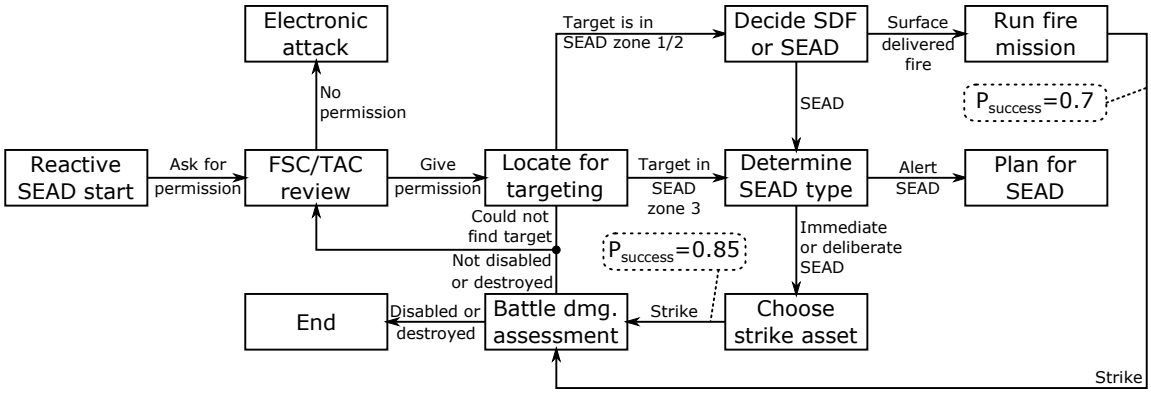


Figure 63: An example OV-5b developed from Marine Corps Warfighting Publication for Suppression of Enemy Air Defenses[60].

the systems of systems is reached. A natural way to think of the activity elements is as states the system is in, i.e., "the system is currently performing function X". This perspective is almost identical to the OV-6b.

The literature has examples of OV-5b models being simulated as colored Petri nets [12] and discrete event simulations [20, 108, 141]. The efforts that are published put a large importance on the dynamic behavior of the system of systems, which makes perfect sense because OV-5b is a behavioral description and the actions it depicts take time to be performed. Because the OV-5b models the activities in an operation, it is highly suitable for modeling efforts representing the dynamics of system of systems behavior such as discrete event, system dynamics, etc. The author has also published work that simulate OV-5bs using discrete event models.

Other modeling approaches such as Markov chains that deal with time could also be used here. It is important to note however, that it is not impossible to think that OV-5b diagrams can be analyzed without resorting to time-based simulations. For example, if one desired to calculate the parallel vs. series nature of the processes in the system of systems a graph representation would suffice or if the probability of accomplishing a certain mission thread were to be calculated, then a conditional probability network can suffice.

The results are summarized in Tables 204–207 given in Appendix A. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a high creation rate (71%) for OV-5b among the projects analyzed [6].

6.6.1 Graph model

OV-5b is a graphical model and as such it can fairly easily be turned into a graph model. However, a graph model of an OV-5b has limited use compared with other models. It can still find cyclic behavior in operations [24], separation of operational activities can provide insight on the timeliness and quality of information received by a later node, and source-sink-transport analysis using the max-flow min-cut theorem algorithms [70]. Given the simplicity of creating graph models, once an OV-5b is available, it is recommended to analyze it with a graph model as a low-fidelity sanity check for the proposed architecture. Given the inevitable combinatorially large size of the architectural design space, rapid models should be used for filtering clearly non-ideal solution away and passing on the very promising design alternatives to higher-fidelity but slow-to-execute models [89]. The results are summarized in Table 44. A hypothetical example of transformation is given in Figure 64.

Table 44: Mapping between OV-5b and graph model elements

	Vertex	Edge
Activity	Y	N
Input/Output	N	Y
Miscellaneous[a]	N	N

[a] Cost, performer, etc.

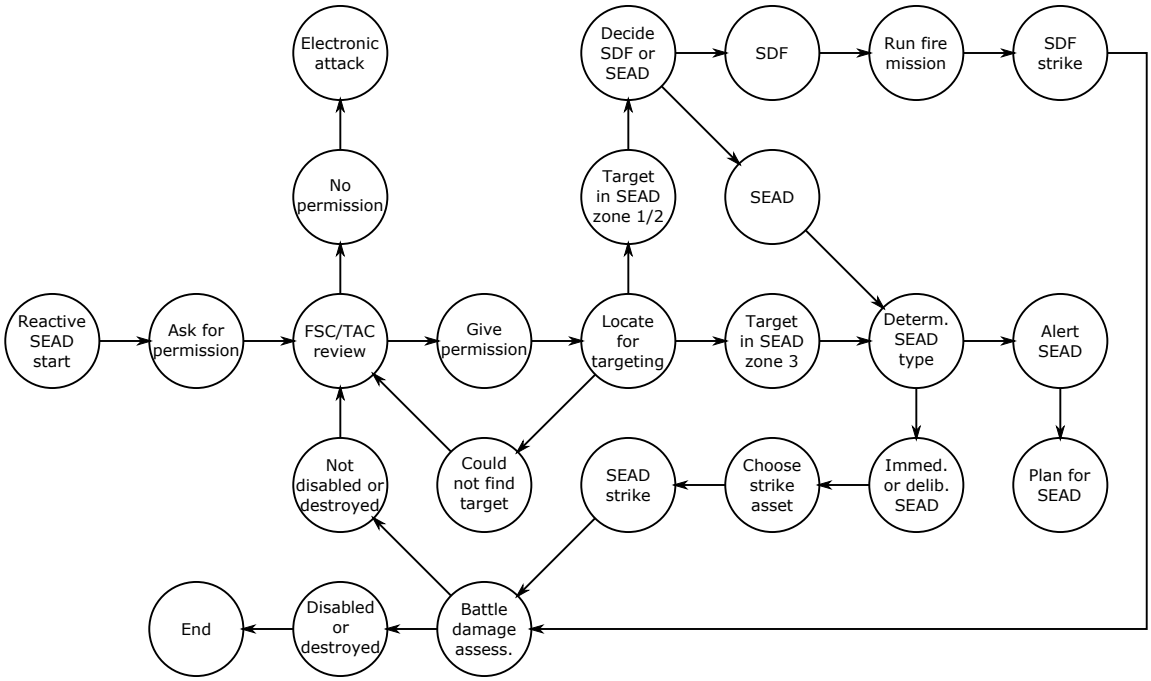


Figure 64: An example OV-5b to graph model transformation based on the OV-5b shown in Figure 63.

6.6.2 Probability model

If conditional probabilities of success for each activity and/or transfer of information from one activity to another, a probability model from the OV-5b can be created. The transformation is fairly straightforward. A hypothetical example is given in Figure 65 where some of the probability metrics come from miscellaneous boxes. Table 45 summarizes the results.

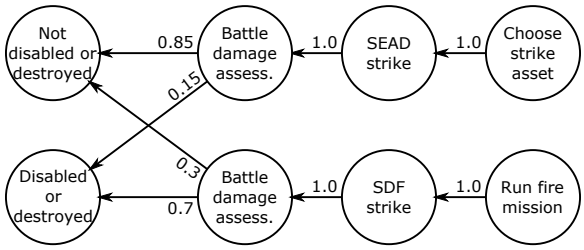


Figure 65: An example OV-5b to probability model transformation based on the OV-5b shown in Figure 63.

Table 45: Mapping between OV-5b and probability model elements

	Conditional probability
Activity	Y
Input/Output	Y
Miscellaneous[a]	M

[a] Cost, performer, etc.

6.6.3 System dynamics model

OV-5b is a fairly good source of information for system dynamics modeling. Each activity node can be thought of as stocks with flows feeding it with scheduled activities and removing finished activities. However, it can be misleading based on how the OV-5b is defined. The reader is reminded that there is no standard way of creating OV-5bs. If two inputs to an activity node means both is necessary for the activity, then system dynamics is not suitable. However, if it means that the input to the activity can originate from multiple other activities, then system dynamics may work. As long as this point is kept in mind system dynamics is a good option to model the system of systems based on its OV-5b. Table 46 summarizes the results. Figure 66 shows an example transformation and the reader can see the extra information required to make the flows work properly.

Table 46: Mapping between OV-5b and system dynamics model elements

	Stock	Flow	Variable
Activity	Y	N	N
Input/Output	N	Y	N
Miscellaneous[a]	N	N	Y

[a] Cost, performer, etc.

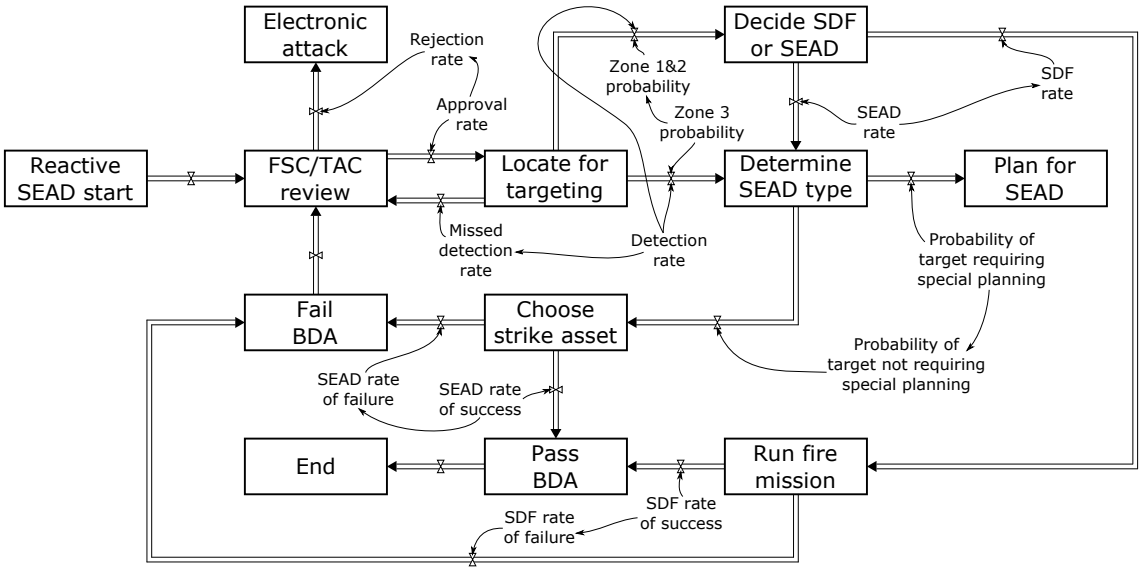


Figure 66: An example OV-5b to system dynamics model transformation based on the OV-5b shown in Figure 63.

6.6.4 Markov chain model

Markov chains can employ OV-5b for a single event's analysis. For example, if the mission is to find and neutralize enemy's surface-to-air missile launchers, a single target can be moved around the Markov chain from states such as: undetected, detected, identified, targeted, shot, neutralized, etc. In this model, one must be careful because multiple targets may be dealt with at the same time and the systems performing the activities may fail the activities or take longer. Markov chains are traditionally used for single system analysis; therefore, the modeler must use caution and judgment. Table 47 summarizes the results.

Table 47: Mapping between OV-5b and Markov chain model elements

	State	Transition
Activity	Y	N
Input/Output	N	Y
Miscellaneous[a]	N	N

[a] Cost, performer, etc.

6.6.5 Petri net model

Petri nets are extremely effective in interpreting OV-5bs into executable models. They have the exact opposite characteristic discussed under system dynamics in which the activity only happens if both inputs to it are true/present. However, the transition can be split into two for each arc the problem will disappear. Therefore, Petri nets are perfectly good matches for modeling the system of systems using its OV-5b description. Additionally, Petri nets do no suffer from the single-system limitation as the Markov chains. If the systems to be modeled are not similar and need to be tracked separately through the OV-5b network, colored tokens can be employed.

AbuSharekh et al. use an OV-5b to shape their colored Petri net model and support it further with OV-6s and a fit for purpose view [12]. Table 48 summarizes the results and Figure 67 shows a typical OV-5b to Petri net transformation. The transformation will vary based on the type of Petri net used (i.e., regular, colored, stochastic, generalized).

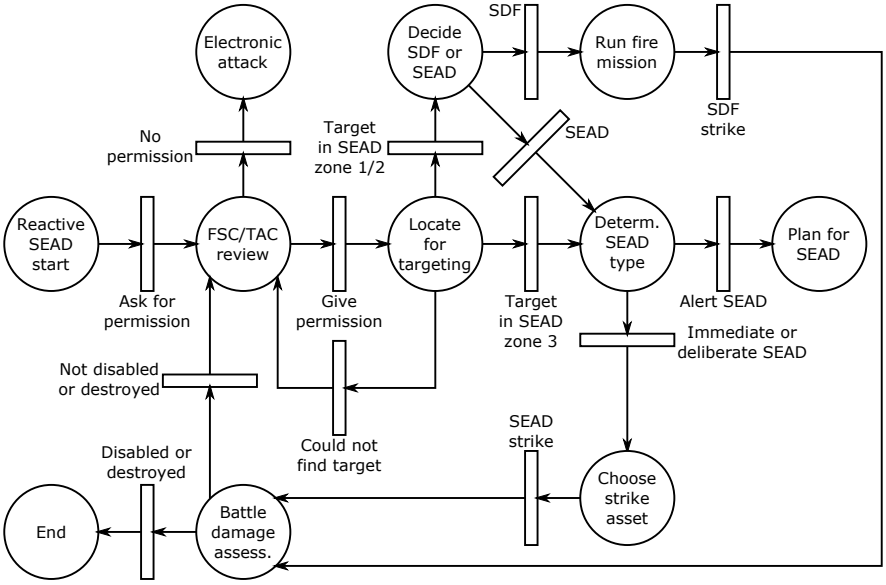


Figure 67: An example OV-5b to Petri net model transformation based on the OV-5b shown in Figure 63.

Table 48: Mapping between OV-5b and Petri net model elements

	Place	Transition	Arc
Activity	Y	N	N
Input/Output	N	Y	Y
Miscellaneous[a]	N	N	N

[a] Cost, performer, etc.

6.6.6 Queueing model

Queueing models are also viable options when OV-5bs available. If the server information is supplied on the OV-5b as performers to activities, a queueing model's skeleton can be set up. The modeler must keep in mind that for all analytical, well-behaved solutions, the arrival rates and processing rates must behave in easily representable random distributions such as Poisson, deterministic, etc. Nevertheless, other than the specific numbers, a queueing model can be constructed from an OV-5b. Table 49 summarizes the results. Figure 68 shows how a part of the OV-5b can be transformed into the queueing theory formalism. "M" denotes a Poisson arrival or Markov process and "D" denotes a deterministic processing time.

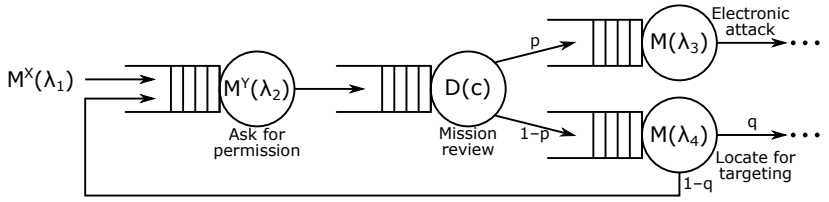


Figure 68: An example OV-5b to queueing model transformation based on the OV-5b shown in Figure 63.

Table 49: Mapping between OV-5b and queueing model elements

	Arrival	Size	Server
Activity	N	Y	N
Input/Output	Y	N	N
Miscellaneous[a]	N	N	M

[a] Cost, performer, etc.

6.6.7 Discrete event model

Discrete event modeling is another very suitable modeling method by using an OV-5b. The events are represented as activities and the input/output lines can be turned into transitions and queues. A hypothetical example is shown in Figure 69. Based on the world view employed for discrete event modeling, the diagram may vary slightly. Similar to queueing models, the servers can be included within the performers of activities. A discrete event model will not suffer from modeling single systems like the Markov chains either. One possible issue that can arise is that the details on entities may be lacking. Additionally, in a competitive/destructive system of systems, the roles of servers or entities may be blurred (e.g., which are entities: SAMs or bombers?). However, for many cases OV-5b includes a significant portion of information needed for discrete event modeling. Baumgarten et al. use the OV-5b as the most detailed piece of operational information in their work and support their models further by system views [28]. Table 50 summarizes the results.

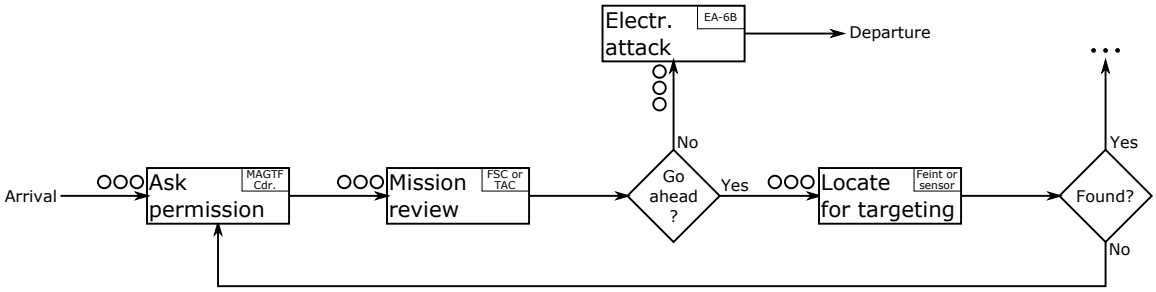


Figure 69: An example OV-5b to discrete event model transformation based on the OV-5b shown in Figure 63.

Table 50: Mapping between OV-5b and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
Activity	Y	N	N	N	N	N
Input/Output	N	Y	Y	N	M	N
Miscellaneous[a]	N	N	N	M	N	M

[a] Cost, performer, etc.

6.6.8 Agent-based model

OV-5b includes information on the actions that would be taken by the agents in an agent-based model. However, it does not necessarily include agents and in the instances that it does, OV-5b does not offer much detail on how the agent performs the activities. It is entirely devoid of environment and agent interaction information. The activities are likely to include some agent rules. Table 51 summarizes the results that OV-5bs are useful in a limited way and they are not a single-view solution for agent-based modeling.

Table 51: Mapping between OV-5b and agent-based model elements

	Agent	Environment	Interaction	Rules
Activity	N	N	N	Y
Input/Output	N	N	N	N
Miscellaneous[a]	M	N	N	N

[a] Cost, performer, etc.

6.7 OV-6a Operational rules model

OV-6a lists the rules under which the system of system operates. The rules vary significantly between one system of systems to another; therefore, all the rules that cannot be represented generally with a graphic fall under the topic of OV-6as. Figure 70 shows operational rules expressed as text in English. Although the text does not follow DoDAF standards strictly, it is a very representative specification for military operational rules. DoDAF only has guidance on the OV-6a if it is written in English (the statements are to follow conditional or absolute imperative forms[59]). But OV-6a rules could be written in pseudocode or logic symbols that can specify conditions and rules more precisely. Figure 71 and Equation 65 show that approach.

$$\forall \tau_i : (\phi_{\tau_i} > 0 \Rightarrow \alpha_{\tau_i} \neq 0) \wedge$$
$$[(\phi_{\tau_i} \leq 0 \Rightarrow \delta_{\tau_i} \neq 0) \wedge (\exists \rho_j, x : x \in \mathbb{R} \wedge \sigma_\tau(\rho_j) = \sigma_\tau(\tau_i) \wedge \sigma(\rho_j))] \quad (65)$$

Both approaches require the definition of variables or symbols. OV-6a is highly useful when modeling detailed rules about the system of systems' behavior and used for modeling purposes surprisingly frequently. There are several examples in the literature using the OV-6a for modeling [12, 108, 141, 190]. In many cases the authors stress the importance of operational rules the OV-6a provides in creating an executable model. However, this view is noticeably more technical than any other operational view that preceded it. Simply put, almost no architecture that was not created for modeling purposes would include an OV-6a. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a low creation rate (32%) for OV-6a among the projects analyzed [6]. The results are summarized in Tables 204–207 given in Appendix A.

Rule 1 When a hostile act occurs or when a force demonstrates hostile intent, use of self-defense is authorized while the force continues to commit hostile acts or exhibit hostile intent.

Rule 2 US warships and aircraft have an obligation to repress piracy on or over international waters directed against any vessel or aircraft, whether US or foreign flagged. For ship and aircraft commanders repressing an act of piracy, the right and obligation of unit self-defense extend to the persons, vessels or aircraft assisted. Every effort should be made to obtain the consent of the coastal state prior to continuation of the pursuit if a fleeing pirate vessel or aircraft proceeds into the territorial sea, archipelagic waters or airspace of that country.

Rule 3 US forces should not enter or remain in areas in which hostilities (not involving the United States) are imminent or occurring between foreign forces, unless directed by proper US authority.

Figure 70: A representative text that can be found in a OV-6a. The three sentences above were taken as examples from Standing Rules of Engagement for U.S. Forces[47].

```
For each MISSILE TRACK entity Instance
  If MISSILE TRACK boost phase code > 0,
    Then MISSILE TRACK acceleration rate is non-null
    Else MISSILE TRACK drag effect rate is non-null
      And
        There Exists a MISSILE TRACK POINT entity instance Such That
          MISSILE TRACK.SOURCE TRACK identifier =
            MISSILE TRACK POINT.SOURCE TRACK identifier
        And
          MISSILE TRACK POINT.SOURCE identifier
  End If
End For
```

Figure 71: A representative pseudocode that can be used as an OV-6a (Reproduced from the work of Mittal et al.[141]). Equation 65 shows the math notation for this pseudocode.

6.7.1 Graph model

OV-6a is presented in text form because graphical forms do not make much sense for rules. Conditional statements can be represented in a graph; however, the contents of the conditionals end up requiring text nevertheless. Even if one were to be able to represent the entire rule structure of the system of systems, the graph model based on that would not simulate the system of systems but its rules, which is not practical. Graph models are simply not suitable for OV-6a views. Table 52 summarizes the findings.

Table 52: Mapping between OV-6a and graph model elements

	Vertex	Edge
Activity	N	N
Relationship	N	N
Rules	N	N

6.7.2 Probability model

Much like the graph models, conditional probabilities do not make much sense when used with a list of rules that the system of systems need to follow. Table 53 summarizes the results.

Table 53: Mapping between OV-6a and probability model elements

	Conditional probability
Activity	N
Relationship	N
Rules	N

6.7.3 System dynamics model

The rules stated in absolute or conditional forms cannot be turned into stocks or flows. Such a representation simply does not make sense. Table 54 shows the totally negative results.

Table 54: Mapping between OV-6a and system dynamics model elements

	Stock	Flow	Variable
Activity	N	N	N
Relationship	N	N	N
Rules	N	N	N

6.7.4 Markov chain model

States and transitions are conceptually very different than rules and do not represent the information given on an OV-6a well. Table 55 details the findings.

Table 55: Mapping between OV-6a and Markov chain model elements

	State	Transition
Activity	N	N
Relationship	N	N
Rules	N	N

6.7.5 Petri net model

Interestingly OV-6as can be represented by Petri net models. Petri net diagrams can express "if α, then β" sentences as shown in Figure 72. Any system with reasonably complicated rules will require many Petri net models (i.e., graphs) to represent its operations. As an example, Rule 1 given in Figure 70 is transformed using a colored

Petri net and shown in Figure 73. The reader can see that even a simple rule can require several states, transitions, and even more arcs. Because a single system will require many Petri net sub-models, detailed models will grow in size very quickly. Both AbuSharekh et al. and Wagenhals et al. use the rules provided in the OV-6a to create colored Petri net models [12, 190]. Petri nets are sometimes used for agent-based modeling to describe the higher lever operation of multiple agents, i.e., to track the states each agent is in (see for example [201]). Table 56 details the results.

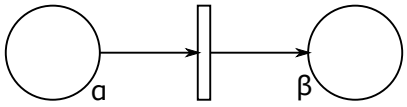


Figure 72: An *if..., then...* statement as a Petri net model.

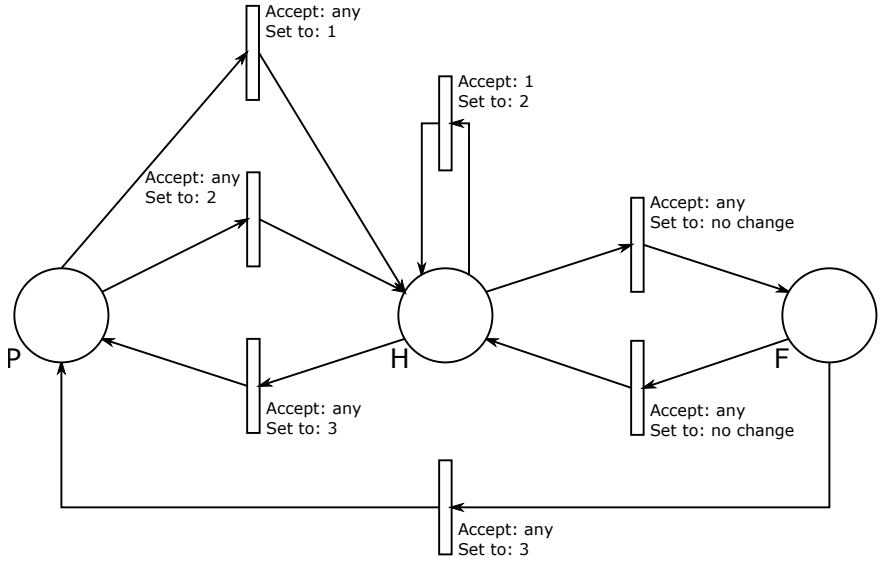


Figure 73: An example OV-6a to Petri net model transformation based on the OV-6a Rule 1 given in Figure 70. Colors 1–3 represent hostile intent, hostile act, and neutral entities. States P, H, and F represent peaceful, hostility, and force use states.

Table 56: Mapping between OV-6a and Petri net model elements

	Place	Transition	Arc
Activity	M	M	N
Relationship	N	N	M
Rules	Y	Y	Y

6.7.6 Queueing model

Some proposed ways of creating OV-6as may possibly be suitable for queueing models, e.g., what happens to jobs after some steps of processing. The reader is reminded that DoDAF does not impose standards on how the views are created and based on the non-specific recommendation, it is best to rule out any reasonable chance that OV-6as can be useful for queuing models. The information provided in an OV-6a has a significant mismatch with the information required to construct queueing models. Most useful rules eventually lead to pseudo-OV-5b or OV-6b artifacts, which is more useful for queueing models. Because rules in sentences do not translate well to queueing, the results are entirely negative as given in Table 57.

Table 57: Mapping between OV-6a and queueing model elements

	Arrival	Size	Server
Activity	N	N	N
Relationship	N	N	N
Rules	N	N	N

6.7.7 Discrete event model

Similar to queueing modeling OV-6a holds little information for discrete event models. Two plausible exceptions exist. The rules may include queue rules such as first-in-first-out, last-in-first-out, priority. Also, OV-6a-listed rules may determine server and entity behavior. Mittal et al. provide an example of how operational rules may be used to increase discrete event model fidelity [141]. They argue that the utility of OV-6a can be increased if a pseudocode format is taken. The findings are summarized in Table 58.

Table 58: Mapping between OV-6a and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
Activity	N	N	N	N	N	N
Relationship	N	N	N	N	N	N
Rules	N	Y	N	M	M	N

6.7.8 Agent-based model

OV-6a is most useful when modeling the system of systems using an agent-based formulation. Due to the nature of agent-based models algorithmic, rule-based behaviors are needed to create agents within the model. Although the rules do not include agent information (such as physical characteristics of agents), they do include information about how the agents must behave. Each agent should be checked against OV-6a rules throughout the simulation for verification. Table 59 summarizes the results. Figure 74 shows where some rules may fit in an agent-based formulation. In the absence of direction by a proper U.S. authority, a U.S. force agent does not enter an area of hostilities not involving the United States. If it receives a direction from an authority, it does enter the area however.

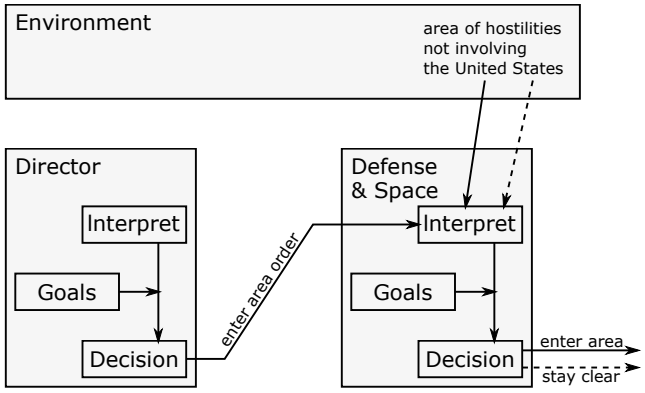


Figure 74: An example OV-6a to agent-based model transformation based on the OV-6a Rule 3 given in Figure 70. The solid lines and dashed lines represent different scenarios.

Table 59: Mapping between OV-6a and agent-based model elements

	Agent	Environment	Interaction	Rules
Activity	N	N	N	N
Relationship	N	N	N	N
Rules	N	Y	N	Y

6.8 OV-6b State transition description

OV-6b is mainly concerned about how the system of systems changes its state. This perspective is very useful especially considering how external events affect the system of systems' operation. OV-6b is very similar to the OV-5b, which shows the order of the activities that constitute an operation and the information they each require, OV-6b goes in detail to show how the system of systems transitions from one operational state to another. The name suggests Markov chains, Petri nets, queueing, and discrete event models may be a good fit in simulating the system of systems described using an OV-6b. In fact the literature has examples of colored Petri net, discrete event, and hybrid modeling approaches that rely heavily on OV-6b views[12, 141, 190].

A usual OV-6b has three elements. The first element is the states that represent the completeness of a job as steps (e.g., student is accepted, passed qualifying exams, has enough credit, proposed thesis, defended thesis, graduated). The second element is the activities that cause a change in the states (e.g., sent acceptance letter, taking qualifying exams, taking classes, submitting proposal document, presenting proposal, writing thesis, presenting defense). Finally, the transitions dictate from which state to which state a system transitions when a triggering activity occurs. Figure 75 shows an example OV-6a. It was adopted from expected enemy behavior from the Marine Corps Warfighting Publication for Suppression of Enemy Air Defenses[60].

The results are summarized in Tables 204–207 given in Appendix A. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations

Report cites a very low creation rate (24%) for OV-6a among the projects analyzed [6].

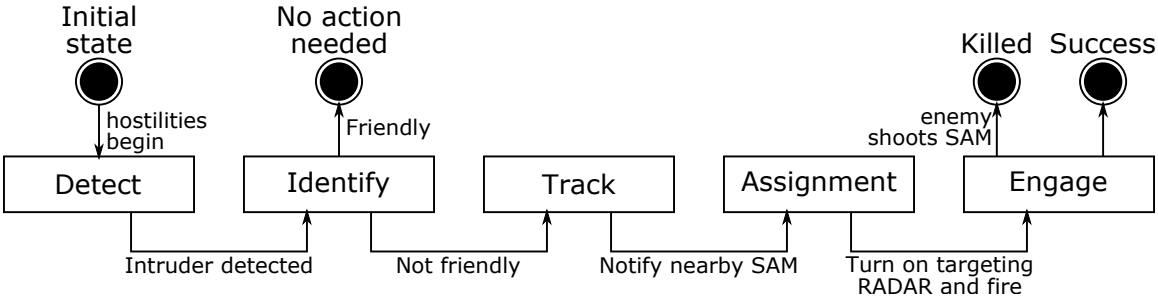


Figure 75: An example OV-6b developed from Marine Corps Warfighting Publication for Suppression of Enemy Air Defenses[60] for how the enemy is expected to operate.

6.8.1 Graph model

A graph model can be constructed from an OV-6b; however, one element must be left out. In a graph model an edge must connect two vertices and only two vertices. An edge cannot connect to another edge or an edge cannot connect more than two vertices. Therefore, two states connected by a transition cannot be connected to an operational activity. A graph model can still be a valuable analysis in finding cyclic behavior in the system of systems' state changing behavior. Table 60 summarizes the results.

Table 60: Mapping between OV-6b and graph model elements

	Vertex	Edge
State	Y	N
Activity	N	N
Transition	N	Y

6.8.2 Probability model

Activities on an OV-6b can be thought of as existing or having a probability to exist. With this type of a formulation a state transition does not necessarily happen, but it has a probability of happening. Similarly, the system of systems only spends a

fraction of time in a state, which can be modeled as a frequency of a probability of being in that state at a given time. Table 61 summarizes the discussion.

Table 61: Mapping between OV-6b and probability model elements

	Conditional probability
State	Y
Activity	M
Transition	M

6.8.3 System dynamics model

OV-6b is an architecture view that includes details about how the system of systems works at a large scale. Each box on an OV-6b can include sub-boxes (i.e., sub-states) that can be used to create a more granular operational state transition description. However, the process of the transitioning is not shown on an OV-6b in any detail-level. The transitions happen instantaneously once the necessary conditions are met. Such a transitioning description is quite useful for system dynamics modeling. A system dynamics model can also deal With unit changes, and if the modeler add to the information included in an OV-6b a system dynamics model can extract very useful insight on the operation of the system of systems described. Table 62 shows the results of the investigation and Figure 76 provides an example transition.

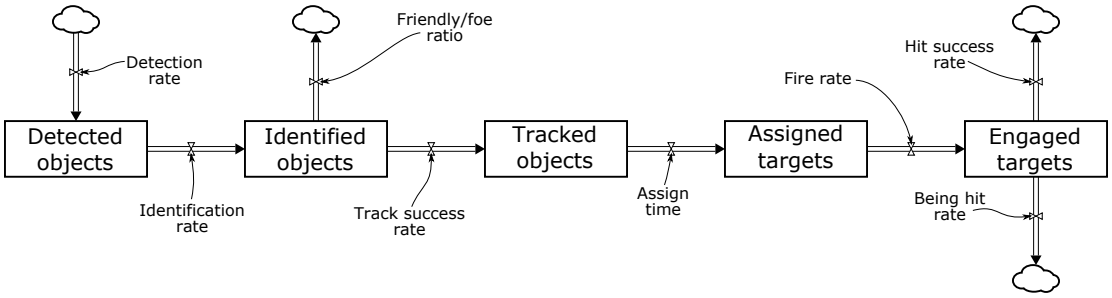


Figure 76: An example system dynamics model translated from the OV-6b shown in Figure 75.

Table 62: Mapping between OV-6b and system dynamics model elements

	Stock	Flow	Variable
State	Y	N	N
Activity	N	N	Y
Transition	N	Y	N

6.8.4 Markov chain model

The Markov chain formulation from an OV-6b resembles the probability model discussion closely. The transitions can either be modeled as probabilities per time step or the rate of change in probabilities in connected states. Either way, an OV-6b carries enough information to create a Markov chain model. Figure 77 shows a hypothetical example from the OV-6b example given in Figure 75. Table 63 summarizes the results.

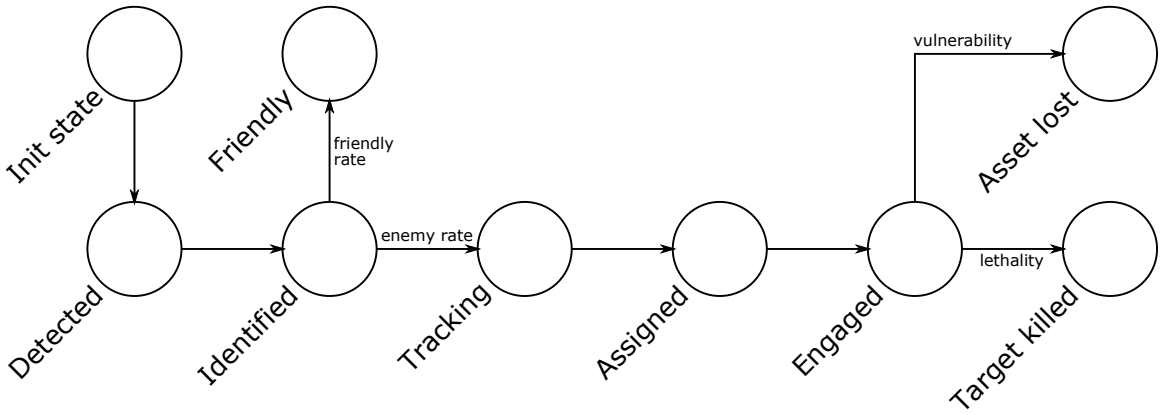


Figure 77: An example Markov chain model translated from the OV-6b shown in Figure 75.

Table 63: Mapping between OV-6b and Markov chain model elements

	State	Transition
State	Y	N
Activity	N	N
Transition	N	Y

6.8.5 Petri net model

OV-6b is a perfect fit for Petri net modeling. The OV-6b states are easily modeled as Petri net places, the OV-6b transitions as Petri net arc, and OV-6b activities as Petri net transitions. Because DoDAF does not prescribe how its views should look like in standard way, one can even draw a Petri net diagram and use it as an OV-6c. Figure 78 shows this similarity and Table 64 summarizes the mapping between elements. The reader can appreciate the simplicity of the translation from the view to the Petri net conceptual model. Xiao-li et al. propose a colored Petri net validation process using UML diagrams that are the basis of the OV-6 series in DoDAF [199]. Similarly, Wagenhals et al. uses the same UML diagrams to construct a fully detailed colored Petri net model [190]. Whereas, AbuSharekh et al. use DoDAF OV-6 views to create and weave more detail into their Petri net models [12].

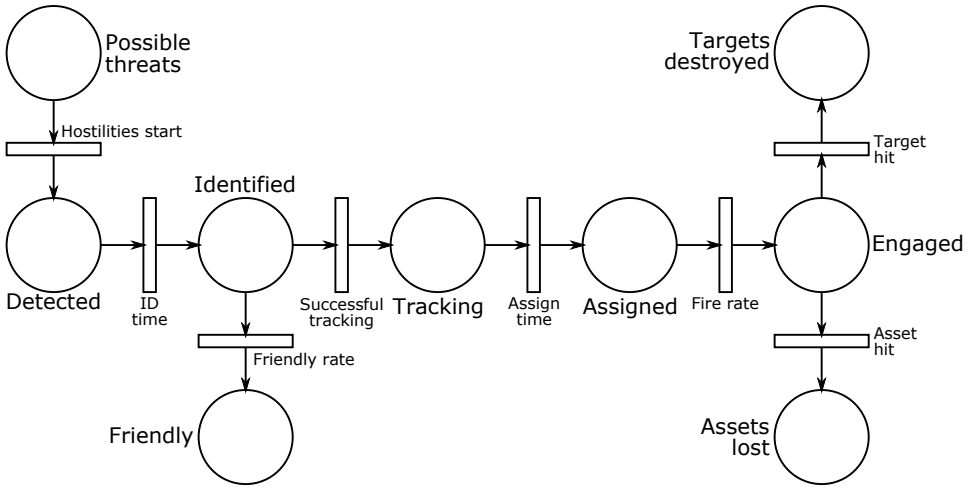


Figure 78: An example Petri net model translated from the OV-6b shown in Figure 75.

Table 64: Mapping between OV-6b and Petri net model elements

	Place	Transition	Arc
State	Y	N	N
Activity	N	Y	N
Transition	N	N	Y

6.8.6 Queueing model

Queueing models can also use OV-6bs to set up a model structure. In the queueing nomenclature the activities would be represented by servers (i.e., the servers are performing the activities). The transitions can be modeled as arrivals to processes, which are states. However, there needs to be some extra information on how difficult it is to deal with the arriving jobs (i.e., how long does it take to process them). Table 65 summarizes the discussion.

Table 65: Mapping between OV-6b and queueing model elements

	Arrival	Size	Server
State	N	N	N
Activity	N	N	Y
Transition	Y	N	N

6.8.7 Discrete event model

Discrete event models are similar to Petri net models and the suitability of OV-6b for discrete event models is not surprising because OV-6b is a perfect match for Petri nets. Optional information that can be displayed on an OV-6b may provide details on how entities are defined and how resources are spent. Apart from that, the mapping is quite similar to the Petri net mapping and is show in Table 66. Mittal et al. use OV-6s as the main source of discrete event modeling structure information [141]. Additionally, Figure 79 depicts a hypothetical translation.

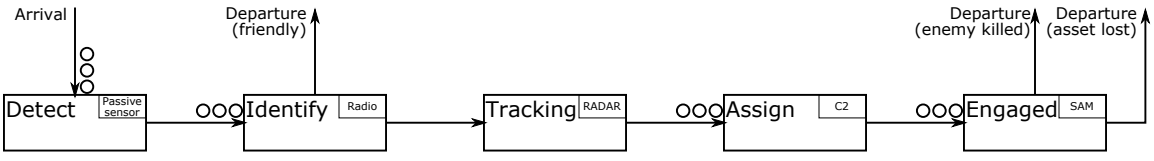


Figure 79: An example discrete event model translated from the OV-6b shown in Figure 75.

Table 66: Mapping between OV-6b and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
State	N	Y	N	N	M	N
Activity	N	N	N	Y	N	N
Transition	Y	N	Y	N	N	N

6.8.8 Agent-based model

Agent-based models require details on each actor's (system, user, or generally: performer) behavior. OV-6b is not a good container for such information. While the elements of an OV-6b match some of the needed information type (e.g., activities may define rules for a system), they lack the detail on how these activities are performed or transitions occur. Combined with the OV-6a, OV-6b could be useful for agent-based modeling however. Table 67 summarizes the results.

Table 67: Mapping between OV-6b and agent-based model elements

	Agent	Environment	Interaction	Rules
State	N	M	N	M
Activity	N	N	N	Y
Transition	N	N	Y	Y

6.9 OV-6c Event-trace description

OV-6c is another highly useful architecture view for modeling purposes. It tracks multiple operational activities over time and shows which are active vs. passive, what event changes their active or passive state, and what resource is passed during the events. Figure 80 shows an example OV-6c representing the use of a coffee machine. Sleepy workers take coffee if it is available. If there is not enough coffee remaining, the next worker brews some more coffee. During the dispensing and brewing, the machine becomes unavailable to others. An OV-6c is valid for a particular scenario; multiple scenarios require multiple OV-6cs. Timing is a critical modeling information that all other discussed operational views lacked so far. OV-6c not only includes

timing but also holds information about concurrency and sequence of activities. The reader can appreciate how this information be useful in dynamic modeling of systems. AbuSharekh et al. highlight the difficulties in dealing with temporal aspect of modeling without the use of an OV-6c [12]. Wagenhals et al. use a UML sequence diagram (roughly equivalent to an OV-6c) to dictate transition timing and definition of scenarios [190]. Xiao-li et al. use UML equivalents of the OV-6c to deal with cases of concurrency, synchronization, and collision within the Petri nets [199]. And finally, as mentioned in the other OV-6 sections, Mittal et al. create parts of the high level operational model structure using the OV-6c[141].

The OV-6c includes the following elements: activities, timelines, and events. The results are summarized in Tables 204–207 given in Appendix A. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a high creation rate (58%) for OV-6a among the projects analyzed [6].

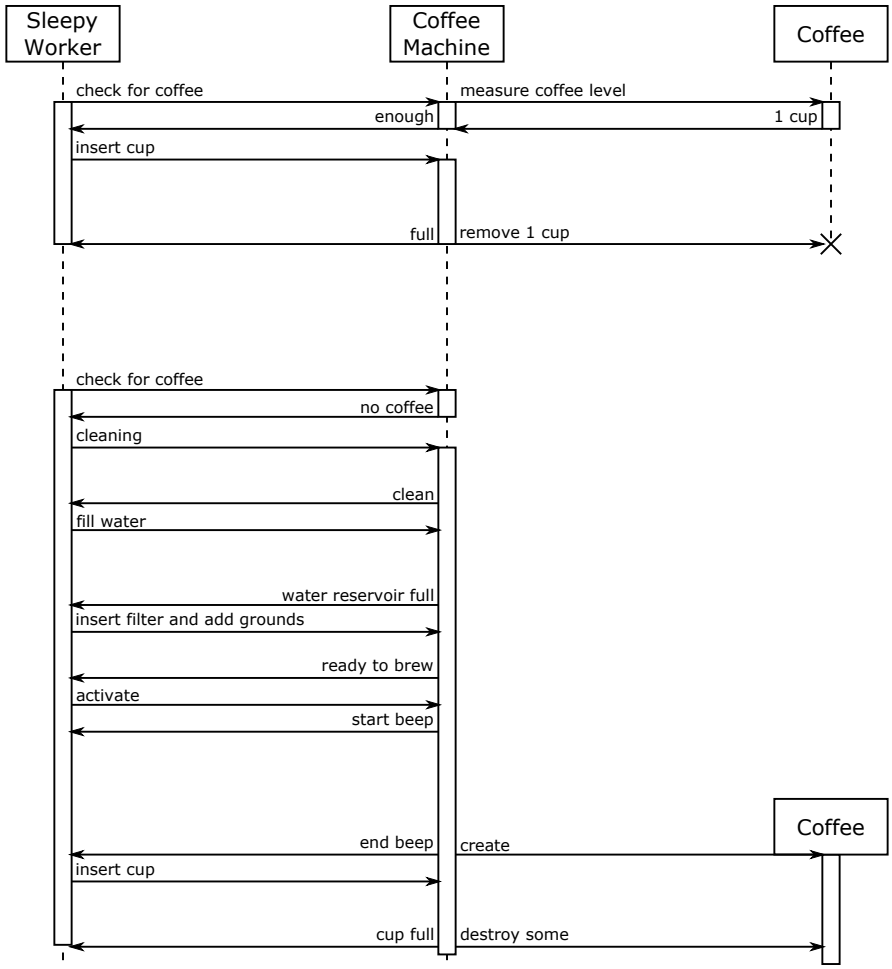


Figure 80: An example OV-6c depicting the operation of a coffee machine in an office.

6.9.1 Graph model

A graph model of a system of systems can be constructed from an OV-6c by simply using operational activities as vertices and events as edges connecting the vertices. In fact, an OV-2 and an OV-5b can be reduced from an OV-6c by following the event between the timelines representing the timing of the operational activities making up the operational nodes and the resources attached to the events are the needlines. The reader is reminded that OV-2 is highly likely to exist alongside with an OV-6c and that using the simpler view instead of the OV-6c is more sensible. Nevertheless, OV-6cs can be used to create graph models of systems of systems. Table 68 summarizes the discussion results.

Table 68: Mapping between OV-6c and graph model elements

	Vertex	Edge
Activity	Y	N
Timeline	N	N
Event	N	Y

6.9.2 Probability model

The events that activate activities on operational timelines can be modeled as probabilities, i.e., they have a probability of happening or executing correctly. For example, an order transmission over radio may fail due to signal interference among other issues with a probability that can be estimated. Based on this discussion, a probability model can easily use the information held in an OV-6c product. Table 6 summarizes the results.

Table 69: Mapping between OV-6c and probability model elements

	Conditional probability
Activity	N
Timeline	N
Event	Y

6.9.3 System dynamics model

System dynamics models as flow of *things* between their respective containers. OV-6c unfortunately does not depict flows or containers but shows how long discrete jobs take to be processed in operational activities. One may find some constants that can be used in system dynamics models in OV-6cs, but more suitable, easier to create architecture views exist that should be used with system dynamics models. The results are shown on Table 70.

Table 70: Mapping between OV-6c and system dynamics model elements

	Stock	Flow	Variable
Activity	N	N	N
Timeline	N	N	N
Event	N	N	N

6.9.4 Markov chain model

For a system of systems that only has a single operational activity going on at any time, a Markov chain model can be created and used accurately. However, if multiple activities are happening concurrently, Markov chain representations would be inaccurate. This is due to the fact that Markov chain represents a single entity's states and the state vector shows the probability of the system being in each of these states. The state vector must sum up to unity and therefore cannot model more than a single entity's probability of being in each of the states. It follows that while OV-6c is not directly very useful for Markov chain modeling, it includes information whether the system of systems violates an assumption that Markov chain formulation relies on and is useful in that highly specific way. Table 71 summarizes the results.

Table 71: Mapping between OV-6c and Markov chain model elements

	State	Transition
Activity	M	N
Timeline	N	N
Event	N	M

6.9.5 Petri net model

Petri net model appear to be a good fit for information presented by OV-6c views at first. There is however a major mismatch in the two approaches: Petri nets do not fire transitions based on time but based on state. Some Petri net formulations can fire based on simulation time (see Wagenhals et al. and AbuSharekh et al. for examples [12, 190]). Their simulation engines however have more in common with discrete event simulations than classical Petri nets. Therefore, the prescribed activity

lengths and activities triggering other activities cannot be modeled using Petri net without roundabout methods which are not ideal. While the transitioning events can be good sources of information for arc, the transition cannot be easily inferred from OV-6c elements. Table 72 summarizes the results. Figure 81 shows an example transformation of the OV-6c shown in Figure 80. Notice that some inhibitors are used; hence, the example is a generalized Petri net. The multiplicity of the bottom right arcs represent that once the transition fires, multiple tokens are placed in the "ready coffee" place.

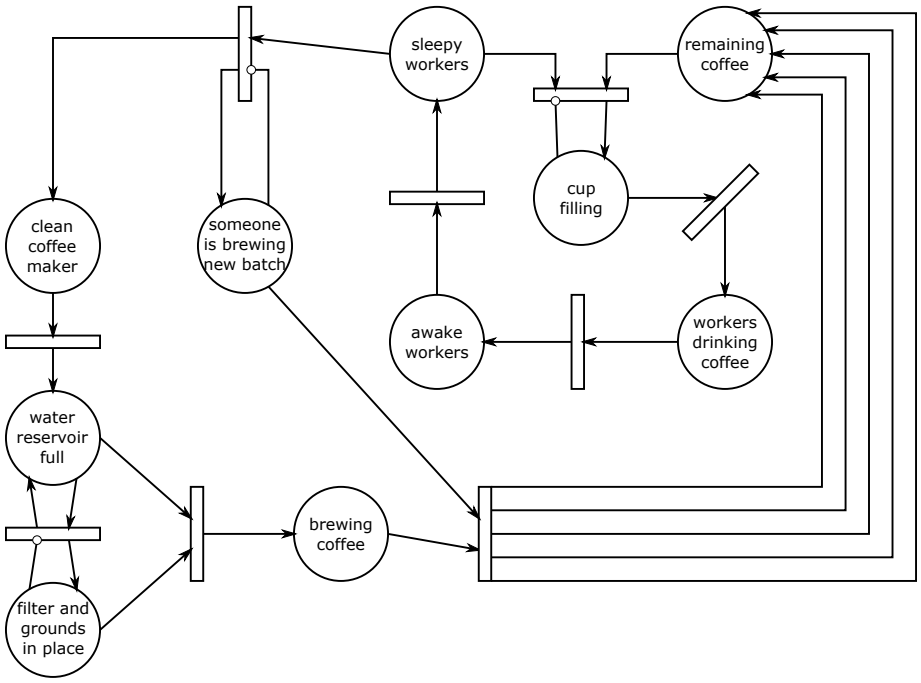


Figure 81: An example OV-6c to Petri net model transformation based on the OV-6c shown in Figure 80.

Table 72: Mapping between OV-6c and Petri net model elements

	Place	Transition	Arc
Activity	Y	N	N
Timeline	N	N	N
Event	N	Y	Y

6.9.6 Queueing model

OV-6c includes a significant amount to timing information, which is important for size of jobs. The events may potentially provide the engineer with arrival intervals; however, it is not in a directly usable format. Additionally, there is virtually no queueing information on an OV-6c. It still includes important information that can be used for modeling a system of systems using a queueing approach. Table 73 summarizes the discussion.

Table 73: Mapping between OV-6c and queueing model elements

	Arrival	Size	Server
Activity	N	Y	Y
Timeline	N	M	M
Event	Y	M	M

6.9.7 Discrete event model

Similar to the queueing models, discrete event models track jobs through events. OV-6c include almost all the information needed. The exceptions are: initial arrival rates, variability in different systems performing the activities if any, and similar minor considerations that need to be determined by the system modeler. These examples demonstrate why it is difficult to automatically generate discrete event models from architectural data. Compared with the lack of information in other view-model pairings, these are mostly insignificant. Mittal et al. propose a modified OV-6a that can fill in most of the aforementioned gaps [141]. Only a small amount of engineer correction is needed to create the model. Additionally, an OV-6c could be a viable option for visualizing simulation runs. The results are summarized in Table 74. A hypothetical transformation based on the OV-6c shown in Figure 80 is shown in Figure 82. Because there is only a single coffee machine as server, only one process can be active at the same time. Coffee is tracked with a variable in addition to the discrete event simulation logic.

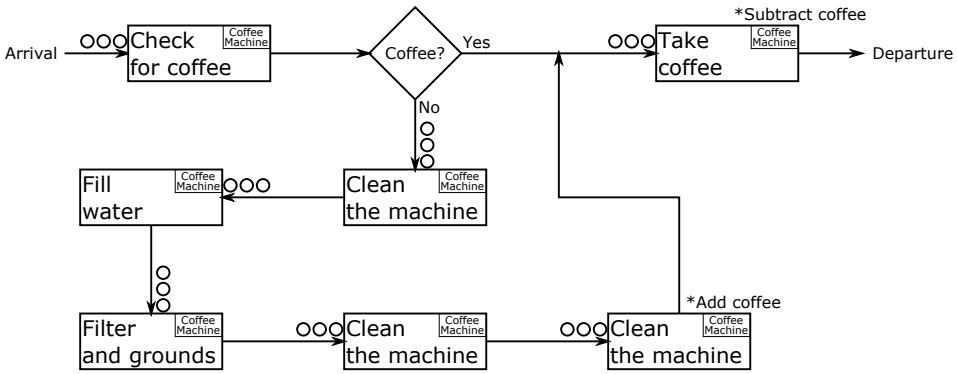


Figure 82: An example OV-6c to discrete event model transformation based on the OV-6c shown in Figure 80.

Table 74: Mapping between OV-6c and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
Activity	N	N	N	Y	N	N
Timeline	N	M	N	N	N	N
Event	Y	M	Y	N	N	M

6.9.8 Agent-based model

Agent-based models require detail on how actors behave as dictated by their internal rules. An OV-6c shows the consequences of these rules. There is valuable information in on OV-6c for agent-based modeling—especially timings and conditions—but they are presented in ways that is not always directly usable. For example, these pieces of information are needed for specific agents but the OV-6c includes only the high-level operational information. Nevertheless, it is the only operational view with detailed timing information. Table 75 summarizes the results.

Table 75: Mapping between OV-6c and agent-based model elements

	Agent	Environment	Interaction	Rules
Activity	N	N	N	M
Timeline	N	N	N	N
Event	N	N	Y	Y

CHAPTER VII

MODELING POTENTIAL OF SYSTEM ARCHITECTURE VIEWS

> Finding the right notation to convey a new concept sounds insignificant [but] a child can multiply 17 by 19 [whereas] the greatest scholars in Rome would have struggled with XVII times XIX [63].
>
> Edward Dolnick

This chapter follows the same format as the previous chapter but deals with DoDAF system views. The discussion at the beginning of the previous chapter also applies here and is not repeated.

7.1 SV-1 Systems interface description

The systems interface description viewpoint is usually based on the OV-2 and fills in the systems that are part of specific operational nodes. However, the main purpose of the SV-1 is to show through what interfaces different resources are transferred between systems. Generally speaking, the resource transfers are the ways different systems interact with each other. The resource exchanges here will satisfy the needlines shown on the OV-2. In other words, they are not part of the needlines but actual technical implementations of resource exchanges that satisfy the needs of each system/operation depicted by the needlines. DoDAF documentation describes the SV-1 as linking "together the operational and systems architecture models by depicting how resources are structured and interact to realize the logical architecture specified in an OV-2[59]".

There may be multiple different exchanges between two systems. For example, a tanker and a fighter aircraft will exchange information through radio antennas but

fuel through refueling ports, or a bomber and a surface to air missile site will interact with each other through radar antennas as well as bombs/missiles. For large systems of systems, it may be desirable to show only a part of these interfaces and interactions in a single view; however, the full set is probably stored in a computer-readable file. Additionally, due to space constraints, the resource exchanges shown on the SV-1 do not include much detail on what exactly is being exchanged and transfer performance. That information is provided by the SV-2, SV-6, and SV-7.

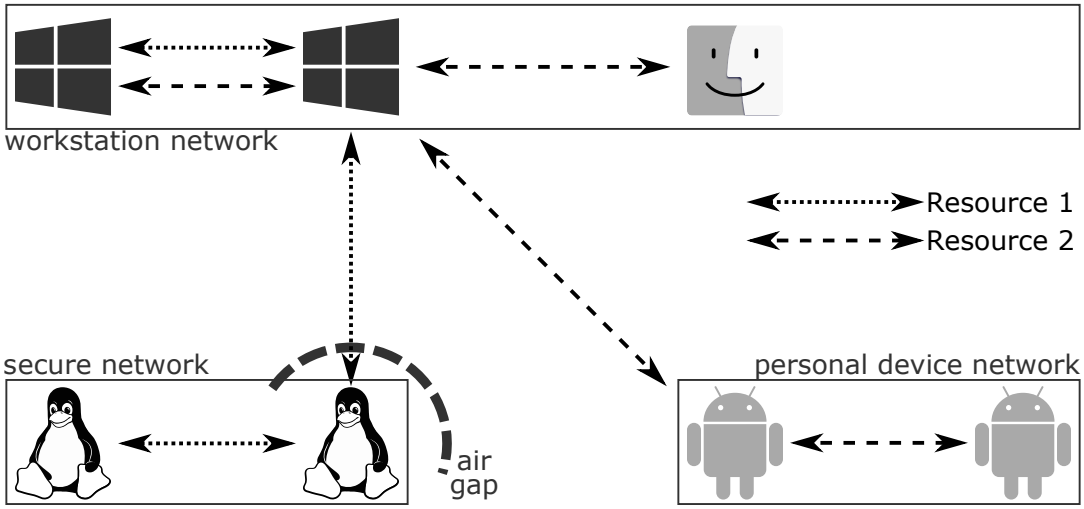


Figure 83: SV-1 reproduced from Jones Wyatt et al.'s example [110] with system node additions

The SV-1 is a commonly developed product. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a high creation rate (71%) for SV-1 among the projects analyzed [6]. An example SV-1 is given in Figure 83 from Jones Wyatt et al.'s work with added system nodes [110]. System nodes were not needed for their work but added to their SV-1 for the purposes of demonstration in this work. They investigate a hypothetical computer network with 2 resource exchanges. The computers within the network are configured to communicate through a subset of four communication methods which are used as the example for the SV-2 later in Section 7.2. Therefore, the view shows systems and interfaces between them. The results from the discussions below are summarized in

Tables 204–207 given in Appendix A.

7.1.1 Graph model

Similar to the OV-2, the graph models fit the SV-1 very well. As discussed above the main idea the SV-1 is communicating is the physical connections between systems through some ports. A graph model is one of the best options to use to represent the SV-1 because mathematical graphs were invented to study the consequences of connections between things. One important difference of SV-1 from the OV-2 is that the system view has three architecture elements instead of two. The extra element is a shell that represents organizations or a parent system and contains systems or sub-systems based on the context. Figure 83 shows a few examples: the different networks the computers belong to.

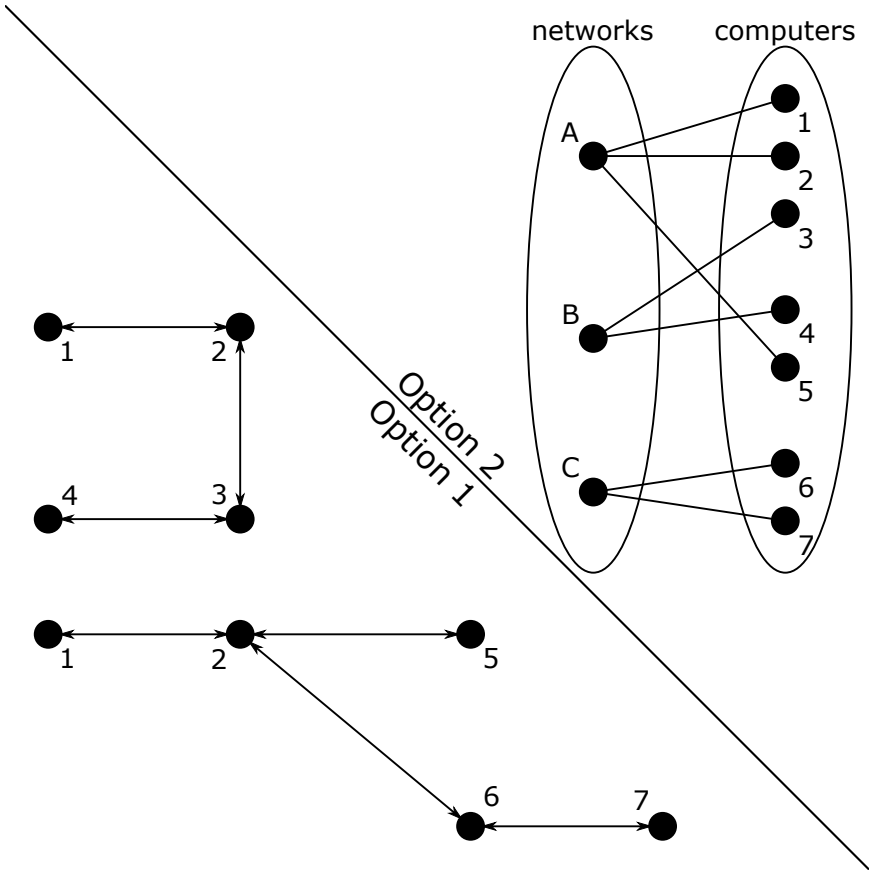


Figure 84: Example SV-1-to-graph model translation options based on Figure 83

The extra element causes issues during modeling that must be solved. A graph model is not fit to handle such inclusion/exclusion statements at the same time as handling connections. The modeler has three options here: completely ignore the shell and just analyze the connectivity between constituent sub-systems, ignore connections and model the membership only, or create two graphs aimed at analyzing connectivity and membership separately. Figure 84 depicts the options. Note that the second option is a special type of graph called bipartite graph. Table 76 provides a mapping between architecture elements to modeling elements for both kinds of graph model created from SV-1s. The systems engineers must remember that they cannot merge the two-types into a single model.

Table 76: Mapping between SV-1 and graph model elements for both connection- and membership-oriented analysis. A single graph model can only be one of the either type, not both.

	Connection type		Membership type	
	Vertex	Edge	Vertex	Edge
System	N	N	N	Y
System node	Y	N	Y	N
Interface	N	Y	N	N

7.1.2 Probability model

There is not much more an analyst can obtain from a probability model that cannot be obtained from a graph model. Simply put, the probability model will be the connection-type graph model with conditional probabilities assigned to the edges. It must be noted that the numerical values for the conditional probabilities will not be found on an SV-1—they may be found on an SV-7—and as such, the SV-1 cannot be converted to a probability model always. The membership type graphs would not have a practical meaning as a probability model. Table 77 summarizes the findings.

Table 77: Mapping between SV-1 and probability model elements

	Conditional probability
System	N
System node	N
Interface	M

7.1.3 System dynamics model

SV-1s can be converted into variable fidelity system dynamics models. When system descriptions are ignored (i.e., only system node and interfaces used), a higher fidelity system dynamics can be constructed for resource exchanges. This model will use the entire detail available on an SV-1. Alternatively, system nodes can be ignored and resource exchanges can be modeled between systems and organizations, resulting in a less-detailed and lower fidelity model that may be more manageable for the entire system of systems. The systems engineer can even choose to include more detail about some systems and less detail about others. Figures 85 and 86 depict an example of variable fidelity using the example from Jones Wyatt's work[110]. While this arrangement is highly flexible, the parameters a system dynamics model needs such as variables, flows, and information about stocks will not be present in an SV-1. Such numbers can most likely be found in the SV-7. Table 78 summarizes the discussion.

Table 78: Mapping between SV-1 and system dynamics model elements

	Stock	Flow	Variable
System	Y	N	N
System node	Y	N	N
Interface	N	Y	N

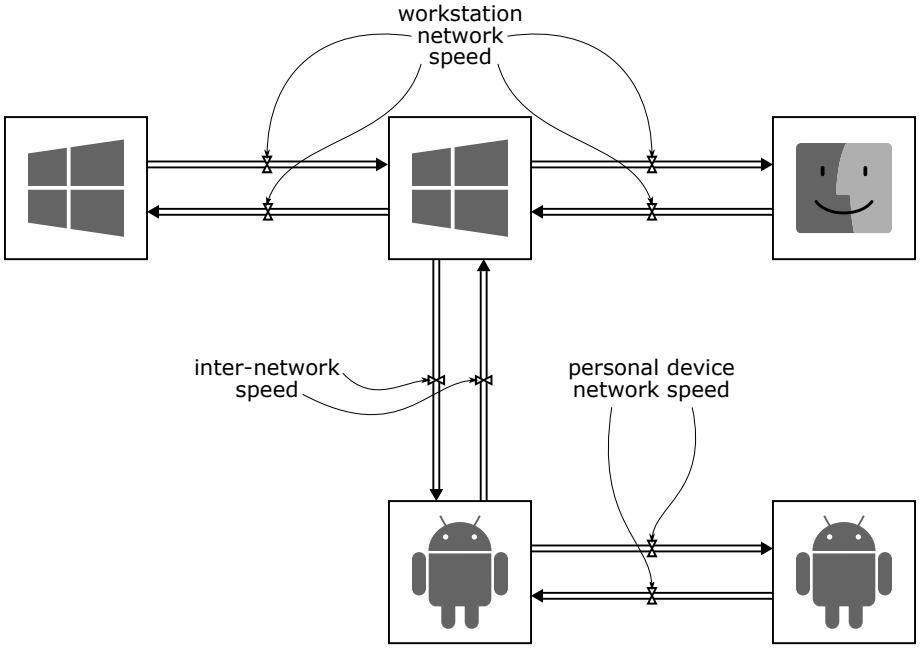


Figure 85: Example high fidelity system dynamics model based on the SV-1 given in Figure 83.

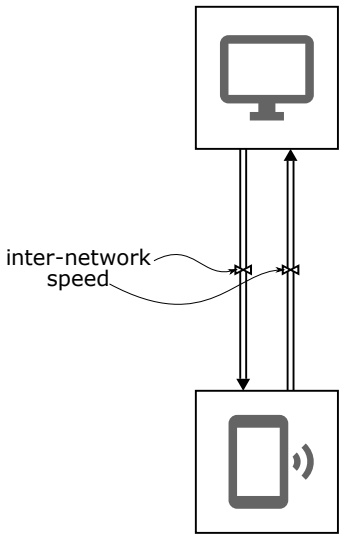


Figure 86: Example low fidelity system dynamics model based on the SV-1 given in Figure 83.

7.1.4 Markov chain model

The SV-1 does not depict any states the systems may be in, it simply shows the systems themselves. Some SV-1s do show functions instead of connections and they can be modeled as transitions in rare cases; however, in general the SV-1 is not a useful view for Markov chain modeling. Table 79 shows the lack of fit between the SV-1 and Markov chains.

Table 79: Mapping between SV-1 and Markov chain model elements

	State	Transition
System	N	N
System node	N	N
Interface	N	M

7.1.5 Petri net model

An SV-1 does not include enough information to create a realistic Petri net model for the system of systems. Petri nets require state specification, transition rules, and some timing information. As discussed in the Markov chain section, states do not exist in SV-1 models. If system nodes can be thought as places and resources as tokens, some kind of Petri net model can be built but the transition rules will be missing in any case. Petri nets are not a good fit for SV-1 models. Table 80 shows the mapping between SV-1 architecture elements to Petri net modeling elements.

Wang and Dagli use two SysML views (the internal block diagram that is similar to the SV-1 and the sequence diagram that is similar to the SV-10c) to create Petri net models [192]. The internal block diagram is used to break the complicated operation of the system into several activities that can be modeled separately in relative isolation from the other activities. While the view is useful, the main source of their information comes from the sequence diagrams. Their work can be used to support the argument here: the SV-1 is not particularly suited for Petri net modeling.

Table 80: Mapping between SV-1 and Petri net model elements

	Place	Transition	Arc
System	M	N	N
System node	M	N	N
Interface	N	N	M

7.1.6 Queueing model

The SV-1 is a high-level overview of how the systems are connected to each other via their interfaces. When used with the SV-2 or SV-6, it adds information to the exchanges by including the way they are realized. However, having only the interface information does not necessarily provide enough knowledge about how the system of systems functions. The exchanged quantities are missing on an SV-1 as it only describes the way such quantities can be exchanged. For example it would specify *road* for the exchange of cars between *buildings*; however, the information on how many cars or what type of cars is missing. Therefore, the information provided here is limited for modeling purposes. Table 81 reflects the fact that while systems and interfaces are identifiable, the information on the actual exchanges are limited.

Table 81: Mapping between SV-1 and queueing model elements

	Arrival	Size	Server
System	N	N	Y
System node	N	N	Y
Interface	M	M	N

7.1.7 Discrete event model

The SV-1 has details on how different quantities of things are exchanged between constituent systems. This description is ultimately important to construct a conceptual model for a discrete event simulation. However, the information provided here is not nearly enough to create an entire discrete event model. The interfaces can be thought of as ways that entities travel from one process to other processes that are performed by systems. The consequences of this arrangement is that systems need

to be split into entities and servers depending on their role in the system of systems. This artificial dichotomy is not a trivial decision for a modeler and blurs the way that the system element within the architecture can be used. Table 82 shows how the system and system node elements are related to many modeling elements. Given the already limited, high-level information provided by the SV-1, it can be concluded that the SV-1 can only play an auxiliary role in discrete event modeling, i.e., an SV-1 alone is not enough to create a model. It can also be argued that the existence of an SV-1 is not a good indication that a discrete event model is suitable for representing the system of systems.

Barnhart et al. investigated positive effects of integrating mission and system simulation using SysML as an intermediating language [25]. They create an OV-1 and an SV-1 and output them via XML using a SysML software solution. They do report that the majority of the parameters the simulation needs were not included within the architecture to simplify its creation. They also discuss the positive and negative consequences of omitting parameters; however, these can easily be stored within an SV-7 separately. They find that the SV-1 was sufficiently detailed to create a discrete event model for their problem and their claim can be taken as a support for the argument included in this thesis with some caveats. Certainly, Barnhart et al.'s problem was conducive for what the SV-1 focuses on describing: communications (a kind of resource transfer). Additionally, even though the translation was carried out algorithmically, a great deal of subject matter expertise and work went into creating the algorithm, which is not problem specific yet admittedly nowhere near general to any system of systems and their SV-1 representations. Keeping these caveats in mind, it can be said that the SV-1 is a useful view for discrete event modeling, and in specific cases, may be enough to create a model on its own with some subject matter expert input.

Another interesting study is Baumgarten and Silverman's use of a handful architecture views to analyze a communications network with a discrete event model [28]. Among the architecture views used is an SV-1. While they do not go into much detail about how the SV-1 was used within their process (they choose to focus on the SV-2), they do report that the SV-1 was used as a transition from the higher-level operational definition into the detailed systems modeling. The conclusions reached here agree with their work as the SV-2 is suitable for discrete event modeling.

Table 82: Mapping between SV-1 and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
System	M	M	N	M	M	N
System node	M	M	N	M	M	N
Interface	N	N	Y	N	M	Y

7.1.8 Agent-based model

The SV-1 includes some information about how two systems are interacting with each other. The interaction itself is not shown but the way it is facilitated is described on the SV-1. This information can be used within agent-based modeling by creating *observe* and *act* functions within agents. Agents will then be taking in what is exchanged via an observation, interpret them, and based on their internal goals, decide to act accordingly. In this formulation the systems are modeled as agents, the interfaces as interactions between agents and rules about them. Figure 87 shows an example translation. Table 83 shows the results.

Table 83: Mapping between SV-1 and agent-based model elements

	Agent	Environment	Interaction	Rules
System	Y	N	N	Y
System node	M	M	N	Y
Interface	N	N	Y	Y

In Chapter 8, while testing one of the FAA architectures, it was determined that the rules do not appear on the combination of SV-1, 2, and 6 in much detail to aid

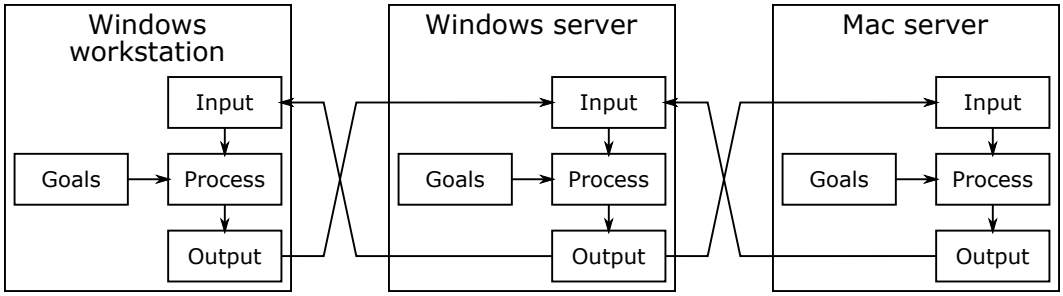


Figure 87: Example SV-1-to-agent-based model translation based on the SV-1 given in Figure 83.

agent-based modeling. Therefore, this map was modified to reflect the fact that rules do not appear on SV-1s. Similar modifications will be seen in SV-2 and 6 sections as well. The modified table is given in Table 84.

Table 84: Modified Mapping between SV-1 and agent-based model elements. The changes were made in light of the attempt at modeling the FAA's Near Term Architecture.

	Agent	Environment	Interaction	Rules
System	Y	N	N	N
System node	M	M	N	M
Interface	N	N	Y	M

7.2 SV-2 Systems resource flow description

The systems resource flow description adds technical detail to the SV-1 interface description. SV-2 shows the means through which the interfacing between systems happen. Historically, this view was used for communications interfacing only; however, with DoDAF2.0 it can be applied to any kind of resource exchange including material transfers. The difference from the SV-1 can be demonstrated best with an example: consider a communication between an F-18 fighter pilot and a commander on an aircraft carrier. The SV-1 will simply put a line between them; however, an SV-2 will add middle steps such as: pilot → communications transceiver → electromagnetic wave → communications satellite transceiver → electromagnetic wave →

carrier transceiver → carrier communications system → network cable → commander. The SV-2 puts the much needed technical detail to the resource exchanges that a modeler would require to build a higher fidelity model. Figure 88 shows an example based on Jones Wyatt et al.'s work [110].

Many elements shown on an SV-2 are not likely to be specific for a single mission. The communications satellite used in the previous example would be used by many different systems performing other missions as well. Many network routers, satellites, the Internet, intranets, etc are such multi-use systems. Similarly, the systems that make up a distribution network for material exchanges are used by multiple users and missions as well. The systems engineer must keep that in mind during the design phase. Distribution networks cannot be designed for a single mission in mind otherwise other missions cannot be completed successfully. As an absurd example, the systems engineer should not assume the entire Internet is there for an airline to sell tickets to customers. Therefore, while the SV-2 is useful for adding technical detail to exchanges between systems, it must include details on how these systems are used by other users and systems for accurate analysis, which may be too much.

Among the modeling types considered in this work, there are three that are traditionally used for communications and supply chain modeling: graphs, queueing, and discrete event. These models will work very well with the SV-2. In some cases the transfer of resources can be modeled as a continuous process rather than a discrete transfer in batches. For such cases system dynamics and continuous-time Markov chains would be useful. The SV-2 is a commonly developed product. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a high creation rate (66%) for SV-2 among the projects analyzed [6]. The results from the discussions below are summarized in Tables 204–207 given in Appendix A.

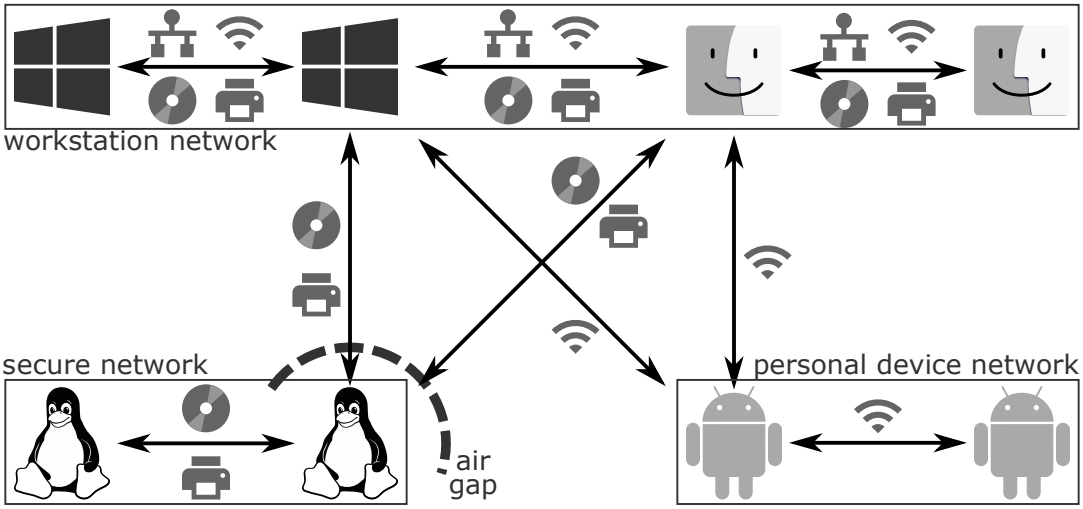


Figure 88: SV-2 reproduced from Jones Wyatt et al.'s example [110] with system node additions

7.2.1 Graph model

The SV-2 is entirely about resource exchanges unlike the SV-1 and a graph model is more easily employable in this case. However, just like the SV-1, the SV-2 also has three architecture elements depicted on it. Consequently, the translation from an SV-2 to a graph model is not as straightforward as the translation from an OV-2 to graph model is. At first, it is natural to think about systems and their ports as graph vertices and connections between ports as graph edges; however, this arrangement also requires an extra edge between a system and its ports as the rules of graphs do not allow two vertices to be connected without an edge between them. While this is not a unsolvable problem, it just adds *phantom* elements to the model that do not exist in the conceptual model as depicted in Figure 89. If the systems engineer wanted to analyze the architecture using weighted edges, these phantom edges require a numerical value that would not exist within the architecture description and must be made up, introducing error and robbing the model of fidelity.

If system and connections between system ports are modeled as vertices, the phantom modeling problem disappears. This arrangement is valid as long as ports

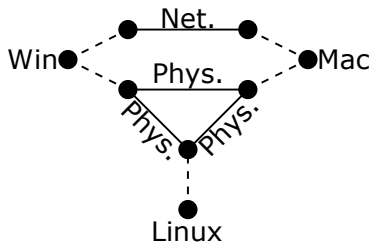


Figure 89: Example SV-2 to graph model translation based on the SV-2 given in Figure 88 that introduces phantom edges. The edges shown with dashed lines do not exist in the architecture but are necessary in the graph model. Print and optical disk exchanges are depicted as *physical* and wired and wireless network data transfers are depicted as *network*.

represented as graph edges always exist between two vertices, meaning that they have only a single connection, i.e., there are no multipurpose ports. If there are such ports in the real architecture, the limitation can be alleviated by splitting multi-purpose ports into separate smaller ports. The formulation still allows for systems to have multiple ports and even allow for modeling bottlenecks caused by connections used by multiple systems at once. Figure 90 shows an example of this translation. The resulting graph model looks significantly different compared to graph models created from OV-2s. Table 85 shows this translation map in a table form.

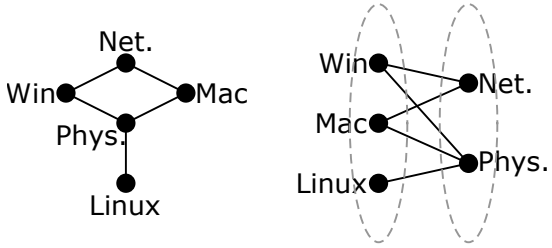


Figure 90: Example SV-2-to-graph model translations based on the SV-2 given in Figure 88 that is free of phantom edges. The graph on the left can be rearranged into the one on the right to highlight the bipartite nature of the resulting graph model.

Table 85: Mapping between SV-2 and graph model elements for the more desirable option without phantom elements

	Vertex	Edge
System	Y	N
Port	N	Y
Flowline	Y	N

7.2.2 Probability model

The graph model from the earlier discussion can be modified to create probability models. If the engineer can assume that the port will not fail (it is a reasonable assumption in most cases), and the failures can only happen within systems or during communication, the previously described arrangement will work perfectly (i.e., systems and exchanges are nodes). Probabilities can then be assigned to each node and the total probability of the operation being a success can be calculated. This can be a very simple but useful model in the early stages of system of systems engineering.

Table 86: Mapping between SV-2 and probability model elements

	Conditional probability
System	Y
Port	N
Flowline	Y

7.2.3 System dynamics model

The SV-2 depicts the exchange of resources and as such system dynamics is a perfect fit to model it. System dynamics deals with stocks and flows of things. Systems can be imagined as stocks of resources and the connections between them as flows of these resources. The ports could be used as variables that adjust flow rates. As the Table 87 shows, there is a perfect one-to-one mapping between system dynamics model elements and SV-2 architecture elements. Figure 91 shows an example transformation. The reader can easily see the perfect one-to-one mapping between architecture elements and modeling elements.

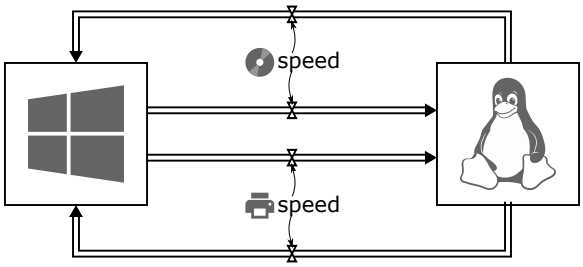


Figure 91: Example SV-2 to system dynamics model translation based on the SV-2 given in Figure 88. Here, only the exchanges between the Windows server and Linux server is shown for clarity.

Table 87: Mapping between SV-2 and system dynamics model elements

	Stock	Flow	Variable
System	Y	N	N
Port	N	N	Y
Flowline	N	Y	N

7.2.4 Markov chain model

A Markov chain model of a SV-2 focuses on where the resources are likely to be found throughout the simulated time and architecture. This formulation is of limited use because it necessarily tracks a single resource rather than large quantities of a resource and many different kinds of resources. While limited, they are still feasible. If different resources need to be tracked, multiple Markov chains can be constructed. Table 88 shows the mapping between architecture elements and modeling elements.

Table 88: Mapping between SV-2 and Markov chain model elements

	State	Transition
System	Y	N
Port	N	N
Flowline	N	Y

7.2.5 Petri net model

Petri nets are a fitting modeling method for representing SV-2s similar to the system dynamics models. Colored Petri nets can track multiple types of resources and track them distinctly. If the system of systems in question is exchanging large quantities

of resources (e.g., vehicle traffic in a city, data transfers on the Internet) a system dynamics model can effectively deal with such large flows, whereas a Petri net formulation would choke. However, a system dynamics model would struggle to keep track of distinct pieces of resources as it tends to aggregate transfers into homogeneous flow of resources. Therefore, the nature of the resource exchanges will dictate which model is more appropriate to model the SV-2. A resource exchange model will not use the entire modeling complexity offered by Petri nets because it will not employ merging and splitting transitions. Table 89 shows the perfect, one-to-one mapping between SV-2 architecture elements and Petri net modeling elements.

Table 89: Mapping between SV-2 and Petri net model elements

	Place	Transition	Arc
System	Y	N	N
Port	N	N	Y
Flowline	N	M	N

7.2.6 Queueing model

The SV-2 provides information about the resource exchanges between the systems and that process can be imagined as the resources within a system queueing up to be transfered out of a port to reach the other end and queueing up to enter the system (the story is highly analogous to international air travel with security checks at either airport). Therefore, the port will specify the way the resource is processed or how much processing it will require. Additionally, the flowline will specify the way the resource arrives to a port and how process-intensive the resource is. Systems ultimately perform the processing and are related to the servers. Table 90 shows the results. It is more complete compared with Table 81.

Table 90: Mapping between SV-2 and queueing model elements

	Arrival	Size	Server
System	N	N	Y
Port	M	M	Y
Flowline	Y	Y	N

7.2.7 Discrete event model

The main difference between the SV-1 and SV-2 from the perspective of discrete event modeling is the inclusion of ports. The port specification allows for peering into the systems and creating multiple events within the system. It must be noted that the way systems are integrated into a system of systems only hints at how the said system of systems operates. The main operation may not require a resource exchange between two systems even thought their ports make that exchange possible for another purpose. Therefore, discrete event models should be built from the SV-5s and SV-10s. The SV-1–4 are mainly there to set the context. Table 91 reflects this point and is suggesting the use of caution.

As discussed in Section 7.1.7, Baumgarten and Silverman use an SV-2 to create a discrete event communications model [28]. While the authors go into detail about their SV-2 views, they also admit that an external source (a narrow-focus SV-7) for many attributes was used to fill in the gaps. The authors also use their subject matter expertise to assigne queues, create activities, and choose what element within the SV-2 are events. As the discussion here suggests, it is possible to create a discrete event oriented SV-2; however, a pre-made SV-2 is unlikely to serve as a good source of information for a discrete event model, because the creator of the architecture will likely not have a strict vision of modeling it later as a discrete event simulation. The SV-2 in Baumgarten and Silverman's work was created very specifically for a discrete event simulation as well as a specific simulation software. The goal of this thesis is to guide a modeler based on a pre-made architecture not to guide an experienced

modeler to create an architecture.

Table 91: Mapping between SV-2 and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
System	N	N	N	Y	M	N
Port	M	M	N	N	N	N
Flowline	N	N	Y	N	M	Y

7.2.8 Agent-based model

With the addition of ports for the exchanges, agents in an agent-based model can be defined better as compared with SV-1. However, in this recipe there is very small difference between agent-based modeling done from an SV-1. Table 92 reflects the similarities (the system and flowline rows are entirely identical, cf. Table 83). The port element from the architecture can map to agent parts as well as the rules about exchanges between the agents.

Table 92: Mapping between SV-2 and agent-based model elements

	Agent	Environment	Interaction	Rules
System	Y	N	N	N
Port	Y	N	Y	Y
Flowline	N	N	Y	Y

In Chapter 8, while testing one of the FAA architectures, it was determined that the rules do not appear on the combination of SV-1, 2, and 6 in much detail to aid agent-based modeling. Therefore, this map was modified to reflect the fact that rules do not appear on SV-2s. The modified table is given in Table 93.

Table 93: Modified mapping between SV-2 and agent-based model elements

	Agent	Environment	Interaction	Rules
System	Y	N	N	N
Port	Y	N	Y	N
Flowline	N	N	Y	M

7.3 SV-3 Systems-to-systems matrix

The system to system matrix is an alternative representation of the SV-1. In realistic architectures, squeezing all system and communication on a single graphic can be virtually impossible. However, when these interfaces are represented as elements of a matrix (i.e., cells in a table), it may be possible to see a large picture of interfaces in a rather small footprint. Basically, the OV-3 matrix summarizes the OV-1 in small-scale architectures and expands it in larger-scale architectures. The matrix can then be used to ascertain all connected systems to each system. The OV-3 is also a more machine-readable format than an SV-1; and therefore, a more useful view for computer modeling. Unsurprisingly, if the modeler does not have an SV-3 to work with, he or she will create one from the SV-1 for modeling purposes. For example, in graph theory nomenclature, the SV-3 is the adjacency matrix of the SV-1 graph. There is virtually no reason not to create an SV-3 if an SV-1 is present.

An architecture may have a layered SV-3 with each layer representing a different kind of interface. This is necessary if the modeler needs to distinguish between resources transferable using specific interfaces. For example, data cannot be transported with trucks or ammunition through radio waves. Unless the SV-3 is layered—or a separate SV-3 is created for each interface type—models will make mistakes by transferring resources on incompatible interfaces. The SV-3 is not a commonly developed product. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a low creation rate (29%) for SV-3 among the projects analyzed [6]. The results from the discussions below are summarized in Tables 204–207 given in Appendix A.

Table 94: Example SV-3 constructed from Jones Wyatt's example problem [111].

	PW	SPW	CDGT	VDGR	CDUT	VDUT	FCS	SP
PW			1					
SPW			2					
CDGT	1	2			1,2			
VDGR		v						
CDUT			1,2				1,2	
VDUT				v				
FCS					1,2			2
SP						v	2	

[1] Command/feedback 1: waypoints, UAV position
[2] Command/feedback 2: pan/tilt/zoom, sensor orientation
[v] Data: video file

7.3.1 Graph model

The SV-3 can be used as an adjacency matrix in the context of Graph models. This means systems will be modeled as vertices and connections between the systems will be represented as edges. This formulation is a simpler model than the one created from the SV-1 and as such, it is of lower fidelity (e.g., it cannot model multiple communications over the same frequency). However, its creation is immediate because the architecture elements do not only map to modeling elements, they are modeling elements. Table 95 details the conversion process with resources as elements of the matrix and systems as rows and columns.

Table 95: Mapping between SV-3 and graph model elements

	Vertex	Edge
System	Y	N
Resource	N	Y

7.3.2 Probability model

The probability model from an SV-3 is a specialized graph model with edge weights as communication/transfer probabilities. The only difference is that the probability

model can be reduced by using the conditional probability rules discussed in Chapter 5. Table 96 shows the mapping between SV-3 and probability models.

Table 96: Mapping between SV-3 and probability model elements

	Conditional probability
System	Y
Resource	Y

7.3.3 System dynamics model

The connections modeled within the systems-to-systems matrix can be represented as flows in a system dynamics model. If there are some values included within the SV-3 matrix, they can be thought of as flow rates (i.e., the rate that resources are being passed between systems). However, the SV-3 is a very simplistic architecture view more focused on summarizing the architecture in a small footprint, and a system dynamics model will be pushing the SV-3s limitations. A system dynamics model would be useful if there are feedback loops, many dependencies between flows and systems. Table 97 shows how the architectural elements of an SV-3 relate to modeling elements of a system dynamics model. It is important to notice that the variable information is likely to be missing from the SV-3.

Table 97: Mapping between SV-3 and system dynamics model elements

	Stock	Flow	Variable
System	Y	N	N
Resource	N	Y	M

7.3.4 Markov chain model

A Markov chain representation from a SV-3 matrix is similar to a graph model. Instead of interpreting the SV-3 as an adjacency matrix, one can interpret it as a transition matrix and the resulting Markov chain can be used to simulate the system of systems. However, the transition matrix must obey some rules discussed

in Chapter 5 but the SV-3 does not. The systems engineer must be very careful in taking the SV-3 as-is and using it immediately in a Markov chain model. Table 98 highlights the similarities between an SV-3 and Markov chain modeling.

Mathieu and Callaway use an OV-4 in problem definition [134]. Their OV-4 can be easily interpreted as an SV-3 or SV-6 with some more specific system definitions. Their work can be used as an example of how an SV-3 can be turned into a Markov chain with some subject matter expert input. The reader is also reminded that the translation was not immediate or obvious; significant modeling knowledge and experience were used to create the model.

Table 98: Mapping between SV-3 and Markov chain model elements

	State	Transition
System	Y	N
Resource	N	Y

7.3.5 Petri net model

A Petri net representation of a SV-3 is essentially a Markov chain but more difficult to simulate. It is recommended that a Markov chain is made from an SV-3 as the architecture view does not offer enough information or elements to match the modeling freedom offered by Petri nets (three modeling elements vs. two architecture elements). Table 99 summarizes the discussion in tabular form. The reader can see that the resource element is used for both arcs and transitions, making the feasible model essentially the same as a Markov chain.

Table 99: Mapping between SV-3 and Petri net model elements

	Place	Transition	Arc
System	Y	N	N
Resource	N	Y	Y

7.3.6 Queueing model

The SV-3 shows how systems are connected with each other. These connections are implemented to facilitate resource exchanges such as data, fuel, passengers, munitions, etc. A queueing model can be constructed to simulate such exchanges as jobs that get processed by the systems and if the system of systems is being developed as a network for resource transfer and processing, e.g., the Internet. However, one must remember that just because there is a connection between two systems, does not mean that there is a procedural hand-off between them in the context of the real operations of the system of systems. Therefore, the results shown in Table 100 are very similar to ones shown in Table 81.

Table 100: Mapping between SV-3 and queueing model elements

	Arrival	Size	Server
System	N	N	Y
Resource	M	M	N

7.3.7 Discrete event model

The discrete event models has many similarities compared with the queueing models; therefore, the discussion is similar. An SV-3 can provide a significant amount of discrete event modeling information if the system of systems is geared towards resource transfer and processing. And just like the queueing model, the systems engineer must remember that a connection between two system nodes does not necessarily mean that they are consecutive steps within a sequential process. Therefore, it is usually not possible to infer the way the system of systems operates from the SV-3. Figure 92 shows an example. Table 101 shows that there is some possibility to create parts of a discrete event model from an SV-3.

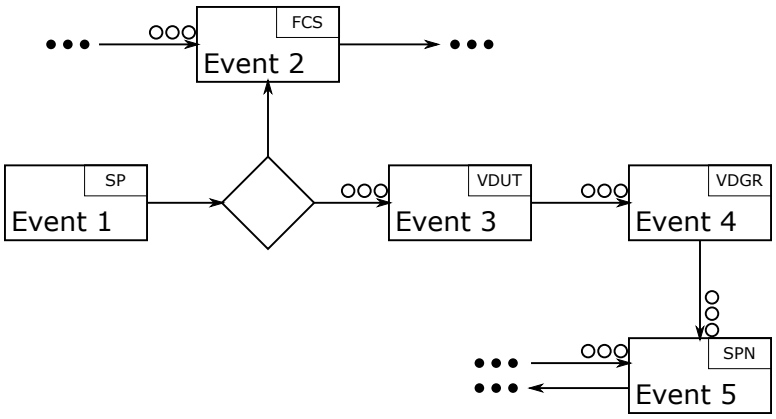


Figure 92: Example SV-3-to-discrete event model translation based on the SV-3 given in Table 94.

Table 101: Mapping between SV-3 and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
System	M	N	N	Y	Y	N
Resource	N	N	Y	N	M	Y

7.3.8 Agent-based model

The SV-3 is a more compact representation of the SV-1 and as such the results are comparable to the ones of the SV-1. It is natural to think of systems as agents within the model, while the resource flows bear the burden of defining interactions and their possible rules. Figure 93 shows an example transformation. Table 102 shows the argument of the SV-3 matrix being too simplistic for agent-based modeling.

Table 102: Mapping between SV-3 and agent-based model elements

	Agent	Environment	Interaction	Rules
System	Y	N	N	N
Resource	N	N	Y	M

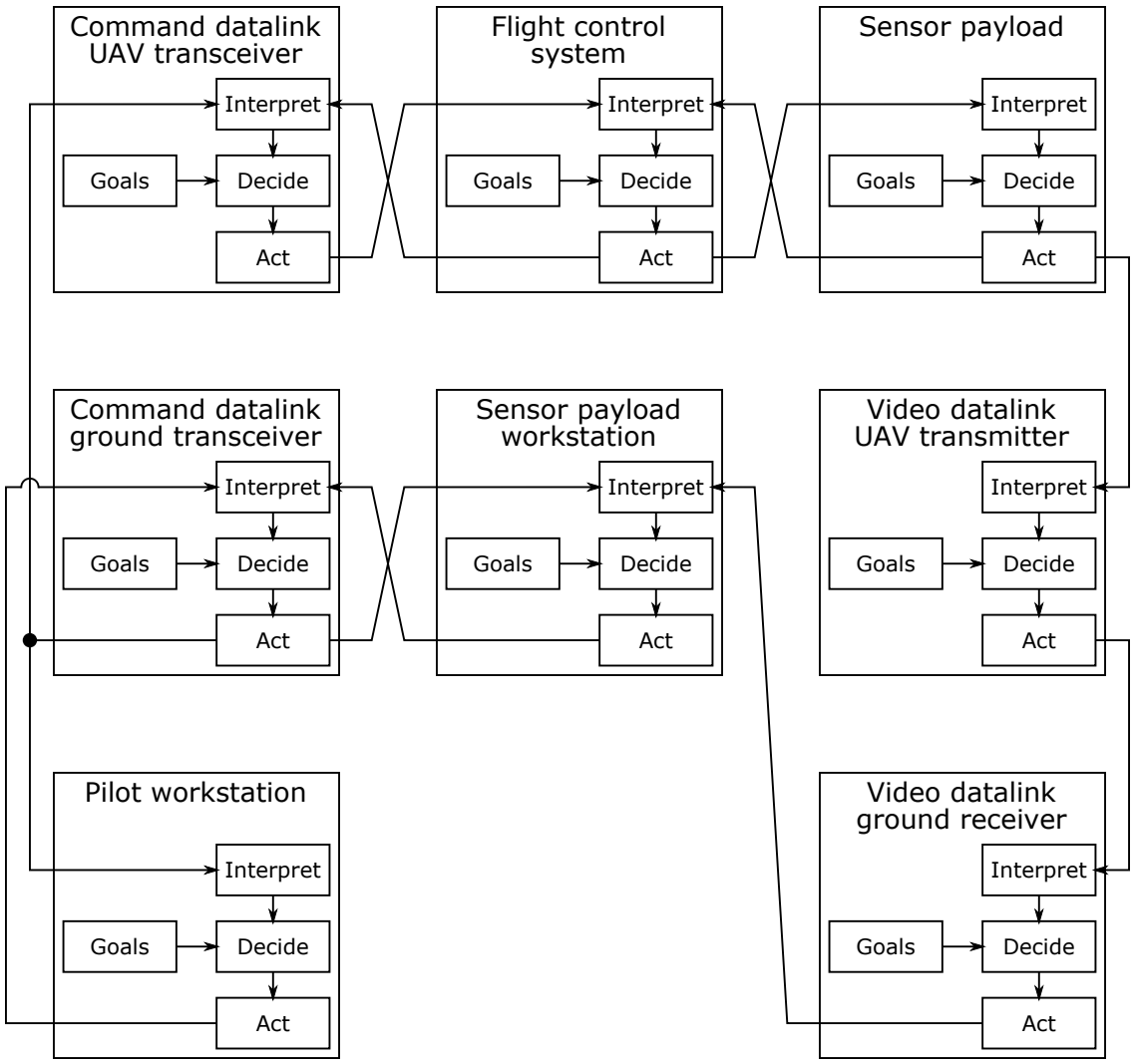


Figure 93: Example SV-3-to-agent-based model translation based on the SV-3 given in Table 94.

7.4 SV-4 Systems functionality description

The Systems Functionality Description is to system views what the OV-5b Operational Activity Model is to the operational views. It includes details on the behavioral aspects of the system of systems depicted on the SV-1. In other words, an SV-4 depicts an intention not the mere existence of an interface. The SV-4 has one difference from the OV-5: stores. It not only shows the functions but also the stores the resources required by the functions. The functions retrieve resources, process them, and

pass them to other functions or resource stores. The SV-4 may not show any systems; its main purpose is to show interactions between system functions. An example SV-4 is given in Figure 94.

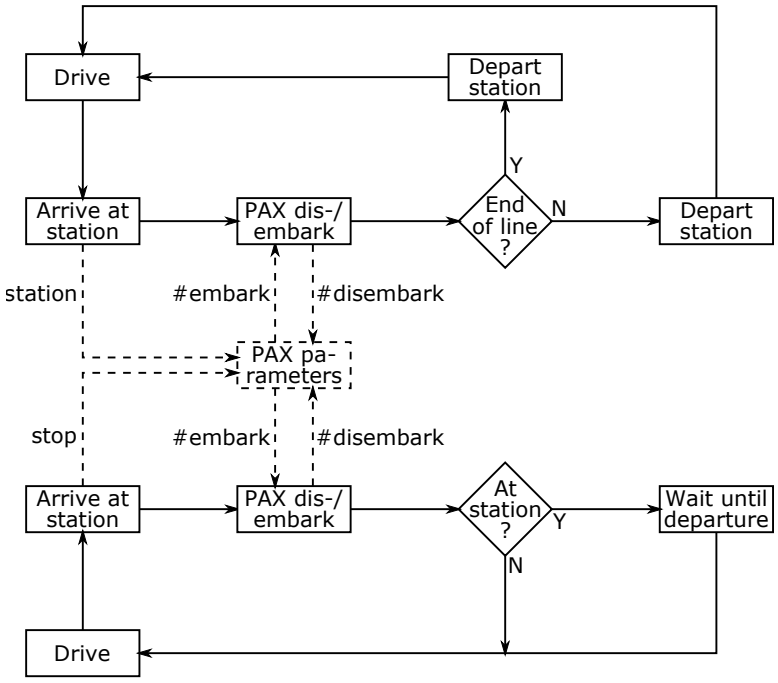


Figure 94: Example SV-4 depicting the interoperation of trains and buses for a public transportation system. Passenger parameters box is a data store and contains all passengers waiting in all stops and stations.

Just like the operational views, the SV-4 has some similarities with the SV-10b and SV-10c. Whereas the OV-10 and SV-10 have specific focus on timing, the modeler must remember that the OV-5b and the SV-4 are more focused on the organization: they do not include order of activities or functions necessarily, but requirements of them. In a sense, the view shows what had to have happened before each function—so it has temporal elements—but lacks concurrency, duration, immediacy, etc. A system of systems engineer can use the SV-4 just like he/she used the OV-5b and the modeling types that support the SV-4 are unsurprisingly similar to that of the OV-5b. The SV-4 is a commonly developed product. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a

high creation rate (55%) for SV-4[1] among the projects analyzed [6]. The results from the discussions below are summarized in Tables 204–207 given in Appendix A.

There is a second version of an SV-4; however, this version does not have a separate name (e.g., SV-4a, it is simply named an SV-4. This second version shows systems functions in a decomposition tree[59]. On a hierarchical SV-4, systems functions can be identified and organized; therefore, this view is practical during architecture development. Unfortunately, a hierarchy of functions is not useful for process simulation and because this work focuses on the modeling and simulation from architecture views, the hierarchical SV-4 is not considered here.

7.4.1 Graph model

A graph model from an SV-4 can analyze a number of architecture characteristics. In the most simplistic analysis, a graph model can be used to determine whether a system function is connected (directly or indirectly) to all the resource stores it needs to work. Additionally, a network distance (number of hops as well as bandwidth considerations) to these resources can be calculated so estimate the lag between a function request and the actual execution fo the function. Another kind of analysis that can be performed with a graph model is the vulnerability of the function via the disruption of resource exchanges. Figure 95 shows an example translation. Finally, Table 103 shows the map between the architecture and modeling elements. Because system functions and data stores cannot be connected in any other way but a function input/output, both can be modeled as vertices in a graph model context without violating mathematical rules. Domerçant uses graphs and algebra to calculate complexities of systems, and the system of systems they belong to [64]. While he does not specifically use an SV-4, his formulation is perfectly applicable to SV-4s.

[1]The SV-4a in DoDAF version 1.5 is equivalent to the SV-4 in DoDAF version 2.

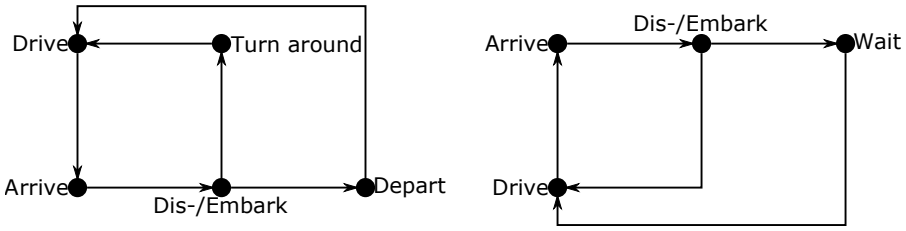


Figure 95: Example SV-4 to graph translation based on Figure 94.

Table 103: Mapping between SV-4 and graph model elements

	Vertex	Edge
System functions	Y	N
Function I/O	N	Y
Data store	M	N

7.4.2 Probability model

A probability model from an SV-4 can be used to calculate overall probabilities for different functions to occur from simpler probabilities that quantify each function's probability if all the previous steps were successful (i.e., conditional probabilities). If the functions are being executed on multiple systems in parallel, the systems engineer can calculate the increased overall probability of functions executing successfully. Table 104 shows how architecture elements can be translated into modeling elements. Only functions are assigned conditional probabilities because the inputs and outputs the functions require are simply the results of previous functions.

Table 104: Mapping between SV-4 and probability model elements

	Conditional probability
System functions	Y
Function I/O	N
Data store	N

7.4.3 System dynamics model

The way the resources travel through the different systems by being processed and changed by their functions can be modeled by a system dynamics model. This model

will represent the functions' inputs and outputs as a flow of information or resources. The flow rate will depend on how fast functions are executed as well as whether previous functions are outputting anything to a function. For example, a very fast function may not be outputting things in large quantities because it may be waiting for an upstream slow function to finish first. This as well as feedbacks that cause bottlenecks can be easily modeled using system dynamics. One issue the systems engineer may run into is that the variables that are ultimately needed to dictate how fast flows work may be missing from the graphical description of the SV-4 and may be included in the SV-7 only. Table 105 summarizes the findings.

Table 105: Mapping between SV-4 and system dynamics model elements

	Stock	Flow	Variable
System functions	Y	N	N
Function I/O	N	Y	N
Data store	Y	N	N

7.4.4 Markov chain model

The SV-4 can be converted into a Markov chain to track the movement of a single resource with a big caveat: Markov chains do not allow splitting a resource to convert it to separate, distinct resources. This is because the Markov model is a probability-mass conserving model, that means the sum of the probabilities of the resource being in all the states must always equal to one. When a split happens in a function (e.g., a computer making a backup copy of the data), this rule will be violated as the total probability will rise to two. Therefore, not all SV-4s can be converted to Markov chains and it would be advisable to use either system dynamics or Petri net models instead. Table 106 summarizes the findings.

Table 106: Mapping between SV-4 and Markov chain model elements

	State	Transition
System functions	M	N
Function I/O	N	M
Data store	M	N

7.4.5 Petri net model

As mentioned in the previous section, a Petri net model is highly suitable for modeling the process the SV-4 represents. A Petri net model can track multiple resource items as well as split, merge, destroy, and create them. This can only happen within Petri net transitions; and therefore, functions on a SV-4 must be represented as transitions. Petri net places only hold tokens, they cannot change them or change their quantities. This means that the function inputs and outputs must be modeled as arcs and states. Such an arrangement seems odd at first because the graphical syntax used in the SV-4 for functions is usually a box. That shape can be more easily associated to circles used for places. Whereas the arrows in the SV-4 graphically are more similar to transitions and arcs. The systems engineer must overcome the urge graphical association and associate elements using their meanings. Figure 96 shows an example translation. Data stores are modeled as places because the resources are stationary in these stores.

It must be noted that system functions can be feasibly represented as Petri net places. However, it comes with added disadvantages. Non-simple transitions such as splits (transition takes in one token but outputs two or more) with mergers (transition takes two or more tokens and outputs one) require *phantom and non-unique* places and transitions to make the model equivalent to the previous arrangement. The phantom elements cause a problem from the perspective of this thesis: they add *maybe* entries into the architecture element to modeling element map. It makes the analysis less crisp and results less definitive. Additionally, one modeling element ends

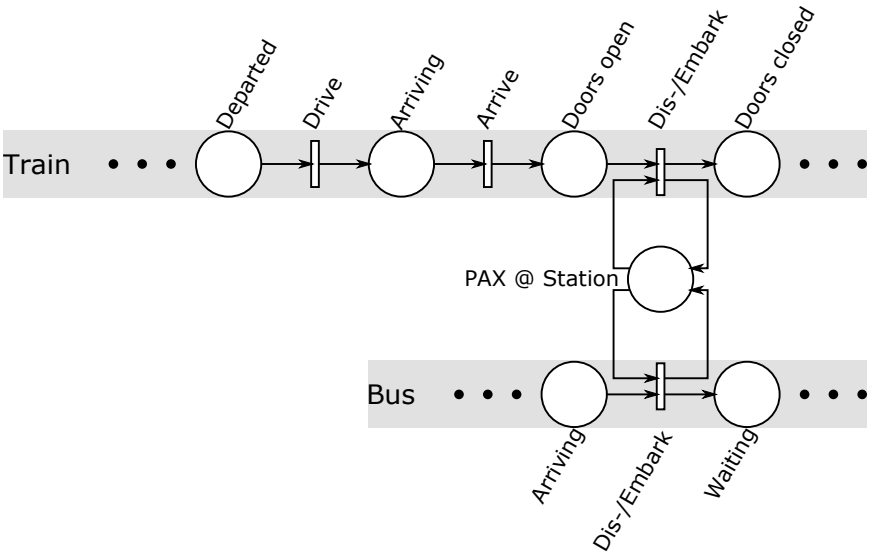


Figure 96: Example SV-4 to Petri net model translation based on Figure 94.

up being mapped to multiple architecture element, which is not desirable as they may include mismatching metrics. Figure 97 shows four options that exist. The generic SV-4 includes a split as well as a merge. The top row shows the author's recommended translation, i.e., a function I/O is represented as a place. The bottom row shows the opposite, i.e., a function is represented as a place. The reader can readily observe in the bottom left cell that with splits that mean a function outputs multiple things at once, and mergers that mean that both inputs are necessary for a function to execute, the modeling elements are mapped to multiple architecture elements. Whereas, the top row has a cleaner mapping.

Table 107: Mapping between SV-4 and Petri net model elements

	Place	Transition	Arc
System functions	N	Y	N
Function I/O	Y	N	Y
Data store	Y	N	N

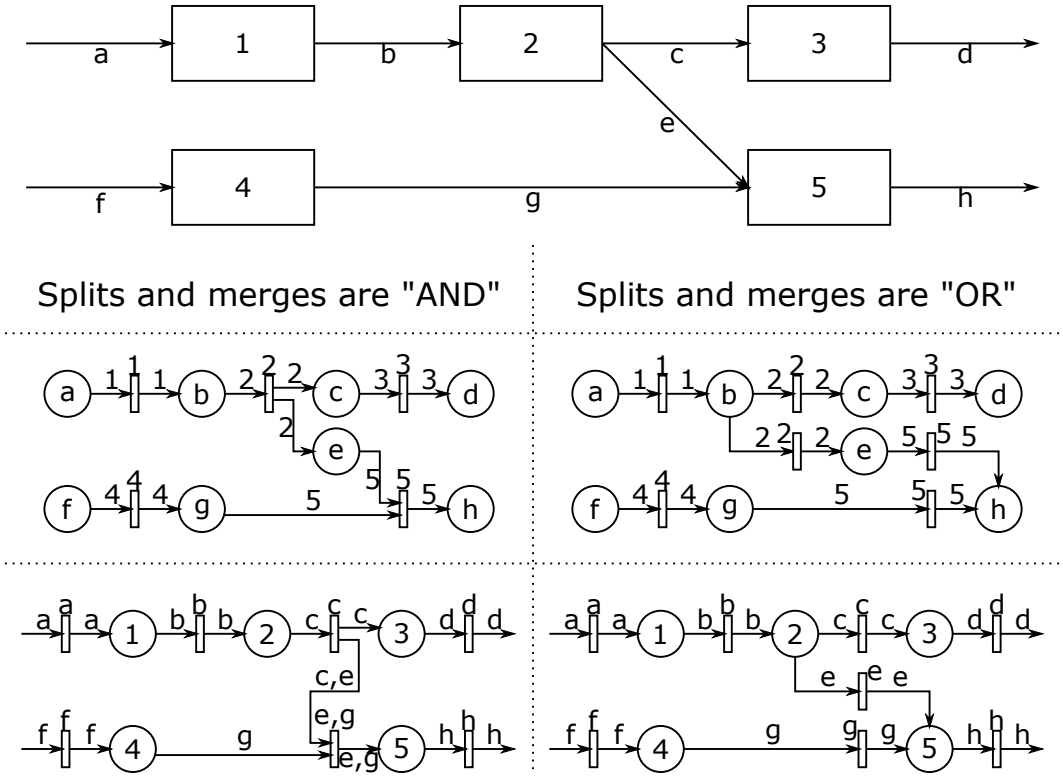


Figure 97: Swapping the places and transitions is possible; however, makes the mapping much less crisp and results in a larger model that is more difficult to simulate

7.4.6 Queueing model

The SV-4 describes what the SV-1–3 lacked: functions in a sequence. Therefore, coupled with any of the previous systems views, it can be used to create a model of the operations in a step-by-step way. Queueing models are used to simulate a network of processes on which different jobs arrive and move until they exit the network as finished; therefore, it is a good fit for modeling out of an SV-4. The data stores can be used as inputs to that network and act as arrivals. The system modeler must use some sequence data from a view such as SV-10b or infer from operational views in which actual order the functions execute. The SV-4 only shows required inputs and outputs of functions. However, the sequences can be changed slightly as long as the requirements for each function are met. Table 108 shows the results of the discussion in a tabular form.

Table 108: Mapping between SV-4 and queueing model elements

	Arrival	Size	Server
System functions	N	N	Y
Function I/O	M	Y	N
Data store	Y	M	N

7.4.7 Discrete event model

A discrete event model will add details and further fidelity to a queueing model. They are very similar; therefore, it is not surprising that both can use the SV-4 very efficiently. The view depicts a step-by-step process that the system of systems operates by and each of these steps can be represented within a discrete event model as events and a server that is employed to perform it. The transitions between these events can be logically mapped to function input and outputs depicted in the SV-4. However, a system modeler must remember that the functions depicted on an SV-4 are organized based on how functions are related to each other not in which order they are executed. While the required discrete event elements are depicted in the SV-4, they are not connected to each other in an executable way. Table 109 summarizes the points in a tabular form.

Mittal et al. propose the use of SV-4 and 5s as one step of 16 to analyze an architecture [141]. Their analysis suggests a discrete event simulation; however, they do not go into much detail other than stating that the SV-4 is used to identify new proposed systems instead of defining already fielded systems. In a different analysis, Mittal argues that the SV-4 does not have activities defined and the design phase in which the SV-4 is created is too early to define components in a discrete event model [142].

Table 109: Mapping between SV-4 and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
System functions	M	N	N	Y	M	N
Function I/O	N	M	M	N	Y	Y
Data store	M	N	N	Y	Y	Y

7.4.8 Agent-based model

The SV-4 is an important source of information for agent-based modeling. It depicts the requirements of the functions that are performed by various systems and users. When defining behaviors of agents, a system modeler can use the information on the SV-4 and lay out the inputs and outputs of an agent behavior. These behaviors may be coded as functions inside a computer model that require precise definition of inputs. Figure 98 shows an example. Based on this formulation, operationally important agent behaviors can be coded in as shells that do not include much logic. That logic will be acquired mainly from the SV-10a as well as the SV-7, SV-10b, and SV-10c. Therefore, while the SV-4 is not enough on its own to create an agent-based model, it establishes a significant portion of the conceptual model that can be coded as a placeholder for further information to finalize later. Table 110 shows the results.

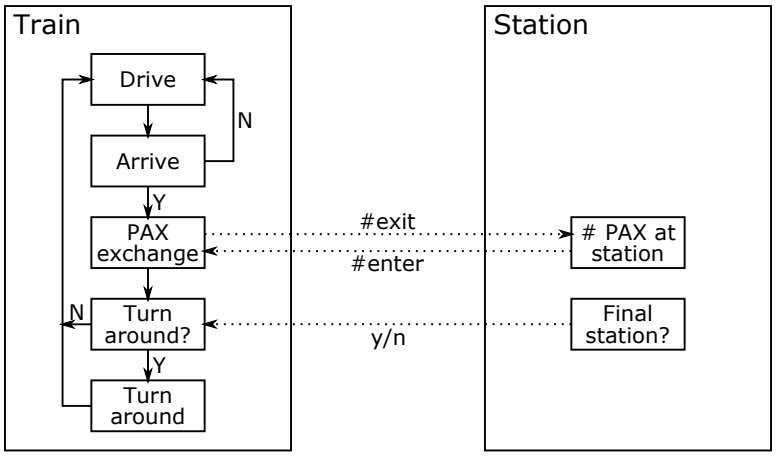


Figure 98: Example SV-4-to-agent-based model translation based on Figure 94.

Table 110: Mapping between SV-4 and agent-based model elements

	Agent	Environment	Interaction	Rules
System functions	N	N	N	Y
Function I/O	N	N	Y	N
Data store	M	M	N	N

7.5 SV-5a Operational activity to systems function traceability matrix

The operational activity to systems function traceability matrix connects the operational aspects of the architecture to the physical systems' functions. With this matrix a systems engineer can see what system functions support what operations. This is a very critical architecture product that connect two other critical views: the OV-5b and the SV-4. It is also a perfect example for how a view alone may not be enough for modeling but invaluable if used with other views as a combination. The only way an SV-5a can be useful for modeling the system of systems is by being used in conjunction with the OV-5b and SV-4. One important note about the mapping is that it does not have to be one-to-one, as multiple functions may support activities and multiple activities may be supported by one function.

An example SV-5a is given in Table 111. Note that the operational activities are linked to the systems via the SV-5b and the system functions are linked to the systems via the SV-4. These links create a useful verification triangle as depicted in Figure 99. The SV-5a is a commonly developed product. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a high creation rate (63%) for SV-5a. The results from the discussions below are summarized in Tables 204–207 given in Appendix A.

Table 111: Example SV-5a for a generic reconnaissance UAV operation

Activity	Function
Sense	Move/fly platform
	Orient sensors
	Activate sensors
	Communicate sensor data
Flight control	Provide trajectory
	Communicate with platform
	Communicate with commander
	Communicate with payload controller
	Monitor platform status
Payload control	Interpret data
	Communicate with flight controller
	Communicate with commander
	Communicate with platform
	Provide orientation to sensors
	Provide points of interest to flight controller

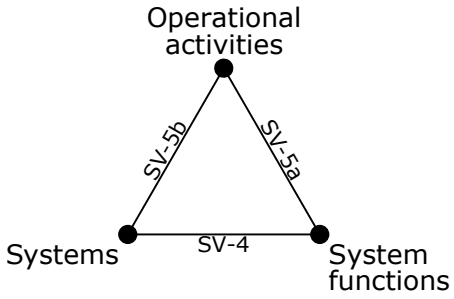


Figure 99: How the SV-4, -5a, and -5b are interlinked

7.5.1 Graph model

The SV-5a is not a particularly useful view for modeling when used alone as discussed above. Therefore, modeling discussions for all the models will be very brief and the results are littered with *maybe* and *no* entries as expected. The systems engineer must always remember to use the SV-5a with the OV-5b as part of a larger modeling effort.

Table 112 shows the most positive result among all the modeling types. Representing both system functions and operational activities as vertices of a graph one can discover central functions that support many activities and in turn identify critical

assets that perform the functions. Additionally, one can discover activities that require an excessive number of functions and argue that they are complicated or tricky to execute. However, these analyses will not be simulations of how the system of systems operates. Nevertheless, they do provide potentially important information. Figure 100 depicts a notional example. The graph model from an SV-5a is a bipartite graph in which one type of node (operational activity) is connected to only the other kind of nodes (system functions) and not to nodes of its own kind.

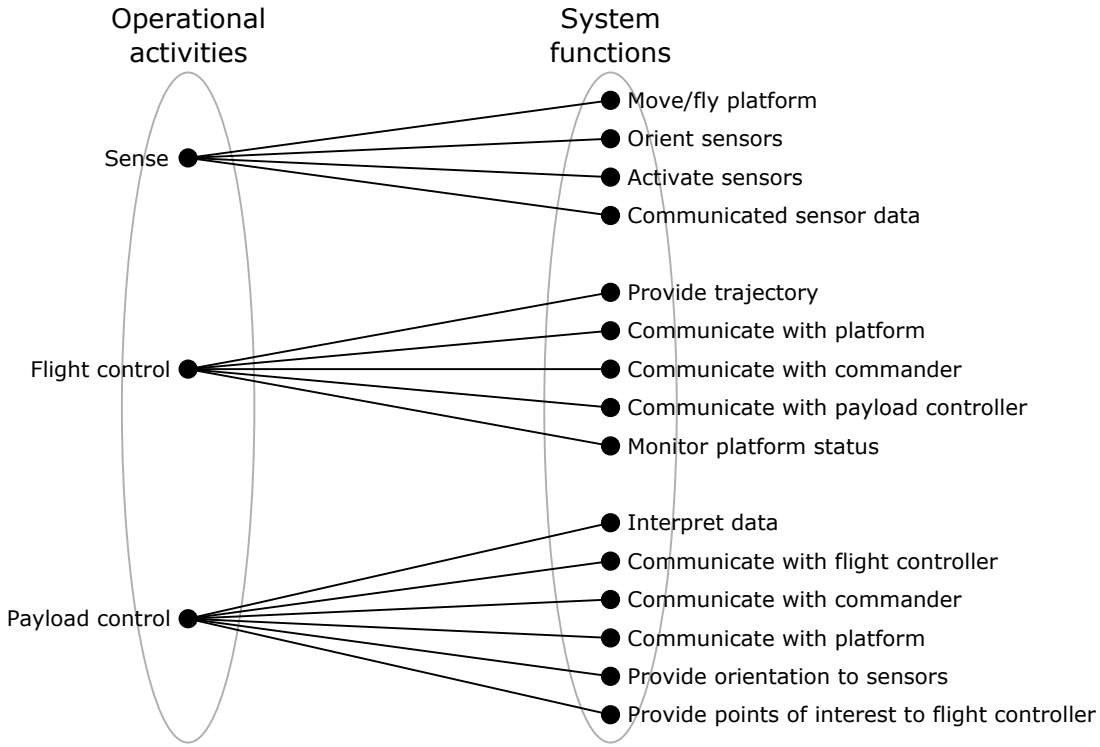


Figure 100: Example SV-5a to graph model based on the SV-5a given in Table 111.

Table 112: Mapping between SV-5a and graph model elements

	Vertex	Edge
System function	Y	N
Operational activity	Y	N

7.5.2 Probability model

The graph model described in the previous section can be modified to include probabilities of functions working properly. When these probabilities are multiplied, an approximation of the operational activity executing successfully can be found. However, the way functions are put together (see SV-4) is also important and a probability model from an SV-5a will ignore the execution structure. Therefore, it is not recommended that the SV-5a be used for probability calculations. Table 113 summarizes the results.

Table 113: Mapping between SV-5a and probability model elements

	Conditional probability
System function	N
Operational activity	N

7.5.3 System dynamics model

The SV-5a holds no resource stock or resource exchange information and is not modelable via a system dynamics model. Table 114 shows this fact in the tabular form.

Table 114: Mapping between SV-5a and system dynamics model elements

	Stock	Flow	Variable
System function	N	N	N
Operational activity	N	N	N

7.5.4 Markov chain model

The SV-5a does not have any elements that can be thought as states and transitions. Table 115 shows the negative arguments in tabular form.

Table 115: Mapping between SV-5a and Markov chain model elements

	State	Transition
System function	N	N
Operational activity	N	N

7.5.5 Petri net model

Petri nets simulate the transformation of things as they change their states. Such information is not offered by the SV-5a. Table 116 summarizes this point in a table. Wang and Dagli use two SysML diagrams (internal block diagram and sequence diagram) to create colored Petri net models [192]. They rely heavily on the information given on the sequence diagrams while using the internal block diagram to split the operation of the system of systems into smaller, more manageable chunks. SV-5as have similarities with the internal block diagrams; consequently, it can be argued that SV-5as are capable of segmenting and layering various other modeling techniques and their inclusion within an architecture does not hint at any particular modeling method being more suitable than others, including Petri nets.

Table 116: Mapping between SV-5a and Petri net model elements

	Place	Transition	Arc
System function	N	N	N
Operational activity	N	N	N

7.5.6 Queueing model

The SV-5a does not provide any information on things that queue up to receive a service from some kind of job server. Table 117 highlights the non-existence of any relation between SV-5a architecture elements and queueing model elements.

Table 117: Mapping between SV-5a and queueing model elements

	Arrival	Size	Server
System function	N	N	N
Operational activity	N	N	N

7.5.7 Discrete event model

Similar to the queueing model, discrete event models also require entities moving through a network of stations where they get some job performed on themselves. The

SV-5a does not hold any such information. Table 118 shows the negative conclusions. Baumgarten and Silverman report that they use an SV-5 along with other architecture views to create a communications model [28]. They do not show or go into detail what was included within this SV-5, while they discuss the SV-1 and 2 in detail. As discussed above in Section 7.5, the SV-5a is not particularly useful on its own, and Baumgarten and Silverman's use fall into this pattern.

Table 118: Mapping between SV-5a and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
System function	N	N	N	N	N	N
Operational activity	N	N	N	N	N	N

7.5.8 Agent-based model

Agent-based models require extensive information about how various systems behave in the system of systems. The SV-5a has no such information. However, the links between the system functions and operational activities can supply the modeler with information on what systems act within what operations. Table 119 shows the results by including *maybe* entries under the rules column. However, the information included within the SV-5a will not be enough for any agent-based modeling.

Table 119: Mapping between SV-5a and agent-based model elements

	Agent	Environment	Interaction	Rules
System function	N	N	N	M
Operational activity	N	N	N	M

7.6 SV-5b Operational activity to systems traceability matrix

The operational activity to systems traceability matrix is similar to the SV-5a but instead of operational activities mapping to systems functions, they map to systems themselves. So, SV-5b connects SV-1 to OV-5a,b. This view's main purpose is to

establish what system participates in what operational activity. Similar to the SV-5a, it is not useful to modeling on its own but virtually every practical architecture would use the information an SV-5b is carrying for modeling purposes. The systems engineer must make sure that each system mapped to operational activities is actually capable of performing that activity. Using the SV-5a, SV-4, and SV-7, these simple checks can be done. Also important to remember is that the mappings do not have to be one-to-one in a similar fashion to the SV-5a. Traditionally, an SV-5b is presented as a table or a matrix. The SV-5b is an product developed not commonly. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a low creation rate (29%) for SV-5b. The results from the discussions below are summarized in Tables 204–207 given in Appendix A.

Table 120: Example SV-5b constructed from author's previous work [20]

Operational activity	Alternatives			
Reconcile target priorities	Commander	Auto. target mgmt. sys.		
Notify carrier of target priorities	Commander			
Remove from target list	Commander	Auto. target mgmt. sys.		
Determine sensor availability	C2 oper.	Auto. asset mgmt. sys.		
Task sensors	C2 oper.			
Confirm target identity	Analyst	Intel sat.	F/A-18	AH-64
	EA-6B	X-47B	RQ-4A	SOF
Manage target movement data	Analyst	Auto. target tracking sys.		
Pass warning/location data	C2 oper.			
Fuse sensor data	Analyst	Data fusion software		
Update target list	C2 oper.	Auto. target mgmt. sys.		
Determine engager availability	C2 oper.	Auto. asset mgmt. sys.		
Assess engagement capability	C2 oper.			
Task engagers	C2 oper.			

.
.
.

7.6.1 Graph model

The SV-5b is equally not effective when modeling the system of systems in isolation. Using the OV-2, OV-5a, OV-5b, SV-5a, and SV-5b the systems engineer can link systems to functions to operational activities to operational nodes to entire operations and this information can be used to support modeling efforts but by itself the SV-5b is not capable of modeling the operation of the system of systems.

One exception is that a graph model can represent the SV-5b in a bipartite graph form as exemplified in Figure 101. The points that can be made are essentially the same as the SV-5a. Systems that are responsible for most operations are expected to be very busy during the operations as they will support the capabilities in many different ways and points during their application. Table 121 summarizes the conclusions.

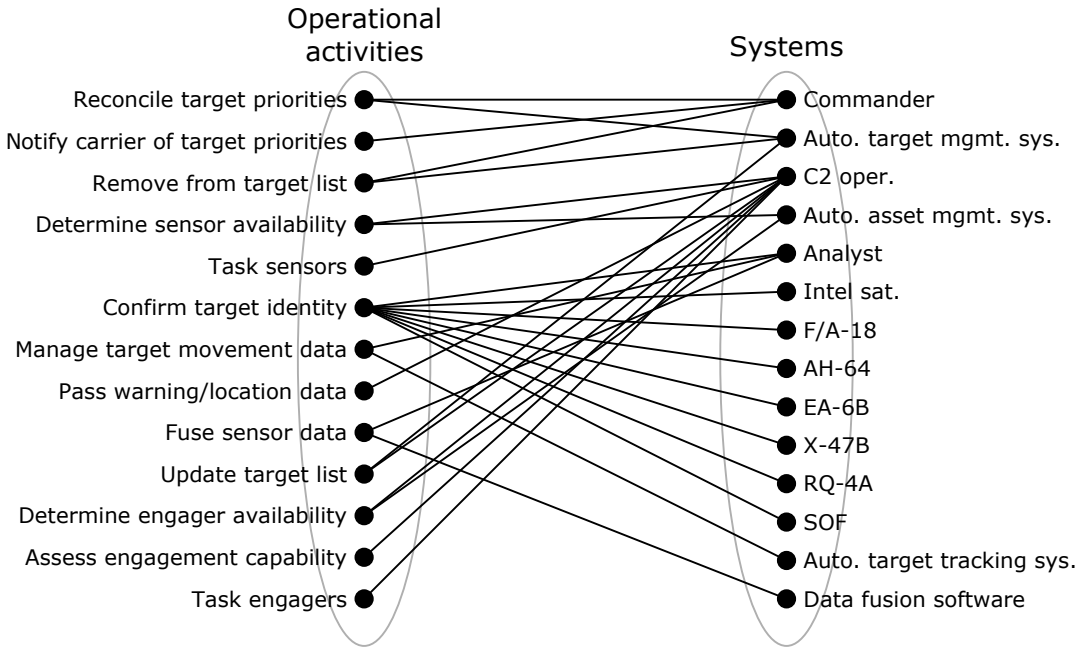


Figure 101: Example SV-5b to graph model based on the SV-5b given in Table 120.

Table 121: Mapping between SV-5b and graph model elements

	Vertex	Edge
System	Y	N
Operational activity	Y	N

7.6.2 Probability model

Similar to the SV-5a, a probability model is not possible to be created from an SV-5b. Refer to the Section 7.5.2 for details. Table 122 shows the negative results.

Table 122: Mapping between SV-5b and probability model elements

	Conditional probability
System	N
Operational activity	N

7.6.3 System dynamics model

The SV-5b holds no information about any kind of resource flow. System dynamics models are not suitable to be used with SV-5bs. Table 123 shows the negative results.

Table 123: Mapping between SV-5b and system dynamics model elements

	Stock	Flow	Variable
System	N	N	N
Operational activity	N	N	N

7.6.4 Markov chain model

Markov chains track the probability of the system being in various states. The SV-5b holds no such information. Table 124 shows the conclusion in tabular form.

Table 124: Mapping between SV-5b and Markov chain model elements

	State	Transition
System	N	N
Operational activity	N	N

7.6.5 Petri net model

Petri nets track things within a network of places and the SV-5b does not have any matching pieces of information to support a Petri net model. Table 125 shows the conclusion.

Table 125: Mapping between SV-5b and Petri net model elements

	Place	Transition	Arc
System	N	N	N
Operational activity	N	N	N

7.6.6 Queueing model

Queueing models examine things that queue up to use a service. The types of information needed to perform that analysis are not included in as SV-5b. A system shown on an SV-5b is performing an activity, which can be seen as a job-server perspective; however, the network is near impossible to be created from the matrix. Table 126 shows the conclusions.

Table 126: Mapping between SV-5b and queueing model elements

	Arrival	Size	Server
System	N	N	M
Operational activity	N	N	N

7.6.7 Discrete event model

The reason that discrete event models are not suitable for SV-5bs is identical to the reason queueing models are not suitable. Table 127 shows the negative results. Baumgarten and Silverman report that they use an SV-5 along with other architecture

views to create a communications model [28]. They do not show or go into detail what was included within this SV-5, while they discuss the SV-1 and 2 in detail. As discussed above in Section 7.6, the SV-5b is not particularly useful on its own, and Baumgarten and Silverman's use fall into this pattern.

Xiong et al. describe a method of evaluating system of systems architectures using a discrete event model [108]. In their work a number of operational views are used. However, their OV-5b includes some extra information that maps system to the operational activities they perform, which is technically an SV-5b in a non-traditional form. Their use is considered as an SV-5b in this thesis. Squeezing extra information into views is a common practice that minimizes the number of architecture views needed to describe a system of systems. Given the extra OV-5b information on top of the SV-5b, it is not surprising that they found their architecture view very useful for discrete event modeling (see Section 6.6.7). However, without the OV-5b information, the SV-5b would not have supplied much information to create a discrete event model.

Table 127: Mapping between SV-5b and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
System	N	N	N	M	N	N
Operational activity	N	N	N	N	N	N

7.6.8 Agent-based model

The SV-5b can be used to determine what operational activity is part of the behavior of what system. Apart from that, it has little value. Table 128 shows *maybe* entries under the rules and negative maps everywhere else.

Table 128: Mapping between SV-5b and agent-based model elements

	Agent	Environment	Interaction	Rules
System	N	N	N	M
Operational activity	N	N	N	M

7.7 SV-6 Systems resource flow matrix

The systems resource flow matrix is closely related to the SV-2 and the systems or physical equivalent of the OV-3. In each row of the SV-6 a description of a resource exchange is given: what is exchanged, what systems exchange the resource, what interface is used, the conditions for the resource exchange, etc. The SV-6 must be carefully developed to cover all operational resource flows. Otherwise, the operation will not be sufficiently supported by physical systems. It is customary to prepare SV-6 tables for exchanges between systems. Exchanges between components/functions within systems are usually scoped out; however, this is not a general rule. Jones Wyatt offers an example [111] in her thesis that is used here to create an SV-6 table shown below in Table 129. The SV-1 is not a commonly developed product. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a low creation rate (47%) for SV-1 among the projects analyzed [6]. The results from the discussions below are summarized in Tables 204–207 given in Appendix A.

Table 129: Example SV-6 constructed from Jones Wyatt's example problem [111].

Source	Sink	Resource
Pilot Workstation	Cmd. datalink ground trans.	Waypoints
Cmd. datalink ground trans.	Pilot Workstation	UAV position
Cmd. datalink ground trans.	Cmd. datalink UAV trans.	Waypoints
Cmd. datalink UAV trans.	Cmd. datalink ground trans.	UAV position
Cmd. datalink ground trans.	Cmd. datalink UAV trans.	Pan/tilt/zoom
Cmd. datalink UAV trans.	Cmd. datalink ground trans.	Sensor orientation
Cmd. datalink ground trans.	Sensor payload workstation	Sensor orientation
Sensor payload workstation	Cmd. datalink ground trans.	Pan/tilt/zoom
Cmd. datalink UAV trans.	Flight control system	Waypoints
Flight control system	Cmd. datalink UAV trans.	UAV position
Cmd. datalink UAV trans.	Flight control system	Pan/tilt/zoom
Flight control system	Cmd. datalink UAV trans.	Sensor orientation
Flight control system	Sensor payload	Pan/tilt/zoom
Sensor payload	Flight control system	Sensor orientation
Sensor payload	Video datalink UAV trans.	Video file
Video datalink UAV trans.	Video datalink ground trans.	Video file
Video datalink ground trans.	Sensor payload workstation	Video file

7.7.1 Graph model

The SV-6 offers more than enough detail to create a graph model. The systems resource flow matrix is a more detailed version the the SV-3 systems-to-systems matrix. The SV-3 only shows the existence of connections between systems, whereas the SV-6 offers details about the connections such as ports, resource types, etc. It is therefore not surprising that a graph model can be created from an SV-6. Table 130 shows the results in tabular form. One difference between graphs created from the SV-3 and SV-6 is that SV-6 includes information about multiple different resources that can be exchanged between systems through various connections; therefore, the resulting graph model may include edges with higher multiplicities. Figure 102 shows three examples, one for each type of resource transfer from Jones Wyatt's work [111]. The source of the model is given in Table 129. The commands and their corresponding feedbacks were grouped together, making three separate graph models out of a single SV-6 table.

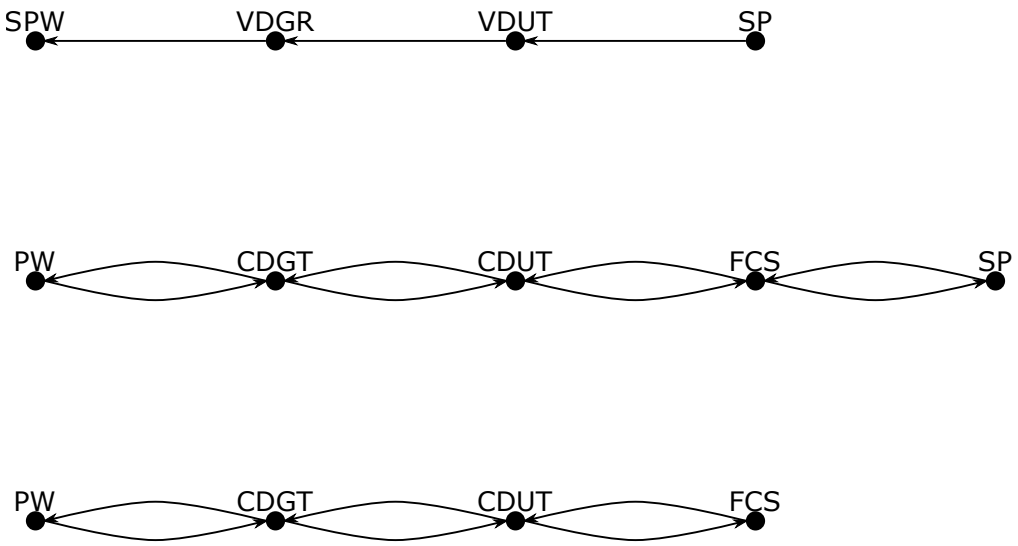


Figure 102: Three example of SV-6-to-graph model translation. The source is Table 129 constructed from Jones Wyatt's thesis [111].

Table 130: Mapping between SV-6 and graph model elements

	Vertex	Edge
System	Y	N
Resource	N	Y

7.7.2 Probability model

A probability model constructed from an SV-6 is essentially the same as one made out of an SV-3. Therefore, the reader is referred back to Section 7.3.2 for details. The only difference is that probabilities for different resource exchanges can be modeled separately without being lumped into a single generic transfer between a pair of systems. Table 131 shows the results.

Table 131: Mapping between SV-6 and probability model elements

	Conditional probability
System	Y
Resource	Y

7.7.3 System dynamics model

The SV-6 provides details about the resource exchanges. In bulk quantities these exchanges can be represented as flows that are the topic of system dynamics modeling as system dynamics ties to model large-scale movements through a system. For example, transferring a video file from a reconnaissance aircraft to a base as a whole cannot be modeled as a "flow"; however, transmitting data at the bits level can be thought of as one. Also, as mentioned earlier, the SV-6 is very similar to the SV-3 and the SV-3 was fairly useful for system dynamics modeling (see Section 7.3.3). For the same reasons, the SV-6 is also useful for system dynamics modeling with the added bonus on details about each exchange. Therefore, the missing information about the variables from the SV-3 is more likely to be included in an SV-6. Table 132 shows the results (cf. Table 97). Figure 103 provides an example.

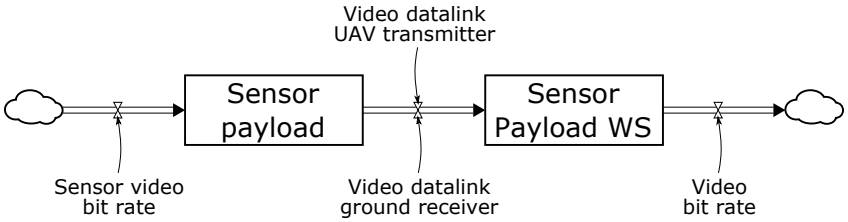


Figure 103: Example SV-6-to-system dynamics model translation based on the SV-6 given in Table 129.

Table 132: Mapping between SV-6 and system dynamics model elements

	Stock	Flow	Variable
System	Y	N	M
Resource	N	Y	Y

7.7.4 Markov chain model

An SV-6 is capable of creating many Markov chain models as resource exchanges are identified as distinct exchanges. Every type of exchange can be represented in a separate Markov chain. The SV-6 also does not have the limitation of the SV-3 for

not following the Markov chain rules. Here the Markov chain will represent the state the resource is in. Therefore, system can be thought as states the resource can be in and the exchanges are how the resources transition from system to system. Table 133 shows the positive results (cf. Table 98). In Section 7.3.4 Mathieu and Callaway's Markov chain model was discussed [134]. The SV-6 is very similar to the SV-3 and the same argument can be used here. Their work can be used as an example of how an SV-6 can be turned into a Markov chain with some subject matter expert input.

Table 133: Mapping between SV-6 and Markov chain model elements

	State	Transition
System	Y	N
Resource	N	Y

7.7.5 Petri net model

The SV-6 can be used as information to build a Petri net model identically as the SV-3 is used to make a Petri net. The SV-6 has mode details and is it possible to create multiple or mode detailed Petri net models. See Figure 104 for an example. Table 134 shows the positive results (cf. Table 99). Bai et al. describe a method for modeling a system of systems using UML as a source for creating Petri net models [199]. In their process, the SV-6[2] and the SV-10c are used as information sources for the Petri net elements. The SV-6 is used as a support to the SV-10b/c and helps with defining the communications aspect of the system of systems. Their work shows what is possible when using multiple views together for modeling and partially supports the position taken here.

[2]The authors refer to an SV-3 in the text but this appears to be a typographical error. Actually, an SV-6 is used, which is evident by the name of the matrix in the text as well as the figure they include for UML-to-DoDAF translation.

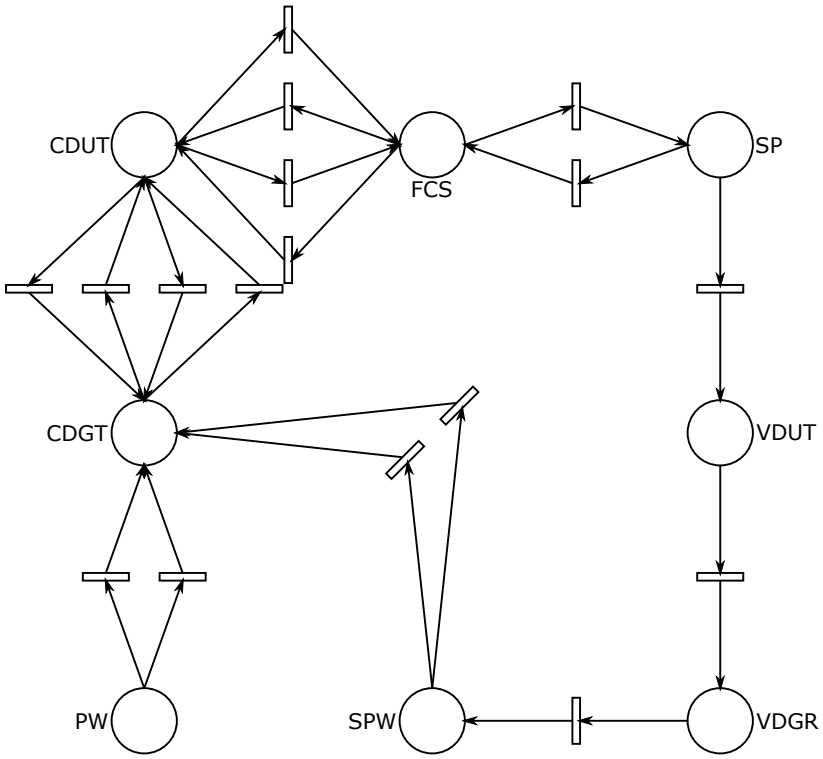


Figure 104: Example SV-6-to-Petri net model translation based on the SV-6 given in Table 129.

Table 134: Mapping between SV-6 and Petri net model elements

	Place	Transition	Arc
System	Y	N	N
Resource	N	Y	Y

7.7.6 Queueing model

The resource exchanges between systems that were defined on the SV-4 provided a sense of how systems could transfer jobs from one to the other. The SV-6 goes further and specifies types of resources exchanged and distinctly list various multiple resource exchanges between systems. The combination of the two provides a clearer view of what is possible with the system of systems. Adding the SV-5a and SV-5b to the mix will certainly be enough to create a queueing model as the real operations are added to the modeler's knowledge about the system of systems. In a way, the SV-4 and SV-6 provide a number of possibilities for a system of systems to perform an operation and

the SV-5a,b paired with some operational views provide the actual way the operation is executed in practice. Table 135 shows promising results.

Table 135: Mapping between SV-6 and queueing model elements

	Arrival	Size	Server
System	N	N	Y
Resource	Y	Y	N

7.7.7 Discrete event model

Similar to the discussion about the queueing models, the SV-6 is highly useful for discrete event modeling. The distinct definition of various resource exchanges between systems allows the system modeler to track different kinds of entities in the discrete event simulation. Different entities can be diverted to different processes that are performed by different servers. The ambiguity of the order of execution remains just as it did in the SV-3; however, with the addition of SV-5s it can be resolved in the same way as described in the queueing model discussion. While the main source of information for a discrete event model comes from the SV-10s (sequence, simultaneity, concurrency, etc.) the SV-4–6 offer a viable alternative for a lower fidelity discrete event model that is certainly feasible. Table 136 shows the results.

Table 136: Mapping between SV-6 and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
System	Y	N	N	Y	M	N
Resource	Y	Y	Y	N	Y	Y

7.7.8 Agent-based model

Agents in an agent-based simulation connect with each other using sensors, communication links, physical transfers, etc. and many of these connections are specified within an SV-6. The matrix is very detailed allowing the system modeler code in necessary inputs and outputs to functions within agent objects which is a significant

part of computer simulation of agent-based models. Therefore, the SV-6 is a highly useful architecture view from an agent-based simulation point of view. The results shown in Table 137 mirror the findings from the SV-2 discussion (cf. Section 7.2.8) as the SV-6 is very similar but more detailed. If the SV-6 is lacking from an architecture, system modeler can default back to an SV-2 to create a skeleton code for an agent-based model. The SV-6 will carry more details and more context in the way the resource exchanges are used and is preferred over an SV-2; however, the SV-2s graphical representation can be convenient to get a grasp on how the agents are put together within the model.

Table 137: Mapping between SV-6 and agent-based model elements

	Agent	Environment	Interaction	Rules
System	Y	N	N	N
Resource	N	N	Y	Y

In Chapter 8, while testing one of the FAA architectures, it was determined that the rules do not appear on the combination of SV-1, 2, and 6 in much detail to aid agent-based modeling. Therefore, this map was modified to reflect the fact that rules do not appear on SV-6s. The modified table is given in Table 138.

Table 138: Modified mapping between SV-6 and agent-based model elements

	Agent	Environment	Interaction	Rules
System	Y	N	N	N
Resource	N	N	Y	N

7.8 SV-7 Systems measures matrix

The Systems Measures Matrix details the measurable characteristics of systems and the resources the exchange with each other. For an air platform the SV-7 may list its range, payload capacity, endurance, resolution of its sensors, amount of information it can downlink, etc. DoDAF manual recommends an SV-7 to be developed simultaneously with the SV-6 to check whether the flows on the SV-6 are realizable. This is

the first of the two uses of the SV-7 for modeling purposes.

The second use is in support of almost all modeling approaches. So far, no architecture view actually listed system capabilities, only the way they are employed. The reader can notice with ease that just because a system is assigned an operational activity to perform, it is not guaranteed that the system will successfully achieve the intended effects. The SV-7 is critical in testing the actual abilities of systems inside simulated scenarios.

For example, using a handful of operational and systems views a suppression of enemy air defenses mission can be modeled. The systems belonging to the system of system can be modeled as modeling elements as discussed before. However, all numerical simulations will ultimately require some metrics, numbers, and measures that are attributed to those modeling elements. In other words, while other architecture views define the structure and operation of the system of systems, the SV-7 is concerned about the performances of the constituent systems (The services equivalent deals with the performance of services). Therefore, in all practical simulation exercises, the values from an SV-7 will be used as inputs to the simulation. And because all models require numerical inputs to function, the SV-7 is a critical architecture view for all modeling types. Some examples are given below.

- An SV-7 may include a throughput for a communication link
  - Inside graph models this value could be used as an edge weight
  - Inside system dynamics models this value could be used as flow rate
  - Inside Markov chain models this value could be used as transition rates
- An SV-7 may include a operation duration
  - Inside a discrete event model this value could be used as a service time

  - Inside a Markov chain model this value could be used as the reciprocal of transition rate

- An SV-7 may include a range for a communication link
  - Inside an agent based model this value could be used as a condition to kill an interaction
  - Inside an discrete event model this value could be used as a failure condition for an event

Because the SV-7 is very useful for many types of models, it is not possible to use the existence of SV-7 in an architecture to make a decision on which model type to use for the analysis of the system of systems in question. The SV-7 may include information more suitable to some modeling types over others however. Therefore, it requires closer inspection in order to select one model type over the other. Additionally, because the SV-7 does not include any information that explains the structure or order of the system of systems, it is not particularly useful in the *conceptual modeling* of the system of systems. The SV-7 is not a commonly developed product. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a almost non-existent creation rate (5%) for SV-7 among the projects analyzed [6]. The results from the discussions below are summarized in Tables 204–207 given in Appendix A.

Table 139: Example SV-7 taken partially from the author's previous work on a system of systems performing a suppression of enemy air defenses mission [20].

System	Action	Metric	Value
Intel Satellite	Discriminate from decoys	Probability of success	0.70
	Assess battle damage	Probability of success	0.70
F/A-18	Discriminate from decoys	Probability of success	0.85
	Assess battle damage	Probability of success	0.85
EA-6B	Discriminate from decoys	Probability of success	0.95
	Assess battle damage	Probability of success	0.95
X-47B	Discriminate from decoys	Probability of success	0.70
	Assess battle damage	Probability of success	0.70
RQ-4A	Assess battle damage	Probability of success	0.80

.
.
.

7.8.1 Graph model

As discussed before the SV-7 does not include any information on how the system of systems is put together. While it is ultimately needed to fill in graph-related metrics such as edge weights, it is deemed that it has no architecture elements that can map to modeling elements. Table 140 shows the results in tabular form.

Table 140: Mapping between SV-7 and graph model elements

	Vertex	Edge
Metric	N	M

7.8.2 Probability model

In contrast to other modeling methods it can be said that the SV-7 provides useful information for a probability model. Because they are built from conditional probabilities that are numbers, the measure definition within the SV-7 could be a conditional probability measure. Table 141 shows the single entry, that is a *maybe*.

Table 141: Mapping between SV-7 and probability model elements

	Conditional probability
Metric	M

7.8.3 System dynamics model

The elements in a system dynamics model require definitions such as what they are connected to and what resource flows between what stocks. Such definitions are not included within the SV-7 measures matrix. Table 142 shows mostly negative results. The SV-7 may include the numeric value for a variable in some cases.

Table 142: Mapping between SV-7 and system dynamics model elements

	Stock	Flow	Variable
Metric	N	N	M

7.8.4 Markov chain model

The measures included in an SV-7 may include some transition probabilities or transition rates for systems to move between their states. The view still lacks information on how these transition metrics are arranged. Therefore, while useful in supporting other views for modeling, an SV-7 is not useful for modeling the system of systems in isolation. Table 143 shows the mainly negative results.

Table 143: Mapping between SV-7 and Markov chain model elements

	State	Transition
Metric	N	M

7.8.5 Petri net model

Similar to the discussion under the Markov chain section, the SV-7 is only possibly useful for defining transition rules and only the numerical part of the rules. Therefore, the SV-7 is deemed of limited use for Petri net modeling as shown on Table 144.

Table 144: Mapping between SV-7 and Petri net model elements

	Place	Transition	Arc
Metric	N	M	N

7.8.6 Queueing model

Queueing model uses many metrics such as allowable queue length, job length, number of jobs arriving with set rates, all of which may be found under the SV-7. However, as discussed several times before, these pieces of information lacks any way of defining the structure of a model but rather sets inputs to the model. As Table 145 shows the SV-7 is determined to be not very useful.

Table 145: Mapping between SV-7 and queueing model elements

	Arrival	Size	Server
Metric	M	M	N

7.8.7 Discrete event model

Discrete event model discussion mirrors the one under queueing model. Some measures listed in SV-7 are ultimately useful for discrete event modeling elements such as server rates, job size, number of entities, and resources spent; however, just like all the previous models, the measures do not prescribe an order to the system of systems. As they can only be counted as inputs to the model, the SV-7 measures are not deemed highly useful for discrete event modeling as shown in Table 146

Table 146: Mapping between SV-7 and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
Metric	M	M	N	M	M	M

7.8.8 Agent-based model

An agent-based model uses large number of measures in order to be executable. Many rules and definitions of interactions do need numerical inputs some of which can be

attributed to geometric logic (such as angles for turning) but some are specific to the systems (a radar system being able to detect targets up to a distance). The latter type may be found in an SV-7. Additionally, in rarer cases[3], number of each system type being employed in a scenario may be included in an SV-7. Table 147 shows the results in tabular form.

Table 147: Mapping between SV-7 and agent-based model elements

	Agent	Environment	Interaction	Rules
Metric	N	N	M	M

7.9 SV-8 Systems evolution description

The Systems Evolution Description shows how the constituent systems and other systems they interact with are changing over time (i.e., evolving). For example, a new system may be added to the system of systems and all the other systems the new one is planned to cooperate with must be *evolved* to work with the new system. This may involve a communications standard change—this would be documented in the StdV-2—or a new antenna or other possible evolutions. The SV-8 keeps a timeline of such changes to the system of systems which is meant to support the evolution in capabilities given in the CV-3. While the evolution over time is useful for modeling the future performance of a system of systems, this information is better represented using *as-is* and *future* versions of all the views discussed so far, i.e., the work would be replicated for the system of systems as it is envisioned to work in the future and working today. The SV-8 is therefore not a particularly useful view for modeling the operations of a system of systems. The SV-8 is not a commonly developed product. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a very low creation rate (24%) for SV-8 among the projects analyzed [6]. The results from the discussions below are

[3]Rarer, because they are not technically system measures

summarized in Tables 204–207 given in Appendix A.

7.9.1 Graph model

While the evolution of a system can be represented as a graph, it is not possible to use that graph to simulate the way that the system of systems works. The SV-8 is a very limited view for modeling a system of systems, a theme that is repeated in the remainder of the modeling types below. Table 148 shows the negative results.

Table 148: Mapping between SV-8 and graph model elements

	Vertex	Edge
Time	N	N
Milestone	N	N

7.9.2 Probability model

The SV-8 has no information regarding probabilities. The probability model is not realizable with an SV-8. Table 149 shows the negative results.

Table 149: Mapping between SV-8 and probability model elements

	Conditional probability
Time	N
Milestone	N

7.9.3 System dynamics model

The SV-8 does not contain information about flows of quantities and their transformation. Therefore, a systems engineer cannot build a system dynamics model from the information included in an SV-8. Table 150 shows the negative results.

Table 150: Mapping between SV-8 and system dynamics model elements

	Stock	Flow	Variable
Time	N	N	N
Milestone	N	N	N

7.9.4 Markov chain model

Markov chains can be effectively used to represent and simulate the evolution of systems in time via upgrades/modernization, spiral development, or replacement. However, this representation and simulation has no relation to the way the system of systems operates. In a sense, the Markov chain network shows the states the current system can be in at different times in the life cycle of the system. For example, when a system is upgraded to a newer version, that system may change its state from Mk1 to Mk2. While interesting and useful in planning purposes, this Markov chain model is not useful in capturing the operations of the system of systems. Therefore, the SV-8 is deemed not fit for modeling the system of systems. Table 151 shows the negative results.

Table 151: Mapping between SV-8 and Markov chain model elements

	State	Transition
Time	N	N
Milestone	N	N

7.9.5 Petri net model

Petri nets are capable of modeling the same concepts from an SV-8 as Markov chains. The points are identical except that Markov chains and Petri nets use different domain specific names (e.g., states vs. places). Therefore, due to the same reason as Markov chains, Petri nets are also determined to be not suitable for modeling the system of systems using an SV-8. Table 152 shows the negative results.

Table 152: Mapping between SV-8 and Petri net model elements

	Place	Transition	Arc
Time	N	N	N
Milestone	N	N	N

7.9.6 Queueing model

A queueing model cannot be constructed from an SV-8 because the SV-8 has no information about queues and performing work on job packets. Table 153 shows the negative results.

Table 153: Mapping between SV-8 and queueing model elements

	Arrival	Size	Server
Time	N	N	N
Milestone	N	N	N

7.9.7 Discrete event model

For the same reason as with the queueing model above, the SV-8 cannot be used to create a discrete event model. It only includes information on how new technologies and new concepts are introduced into the system of systems over time. It may include expected performance or process improvements but such details are usually documented within *current*, *near-term*, and *far-term* future architecture views for all other views. Therefore, the use of an SV-8 is very limited for modeling. Table 154 shows the negative results.

Table 154: Mapping between SV-8 and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
Time	N	N	N	N	N	N
Milestone	N	N	N	N	N	N

7.9.8 Agent-based model

The SV-8 has no information on how the systems behave, how they communicated, how they are operated, or what kind of environment they are operating within. Therefore, the SV-8 is not a particularly useful view for agent-based modeling. Table 155 shows the negative results.

Table 155: Mapping between SV-8 and agent-based model elements

	Agent	Environment	Interaction	Rules
Time	N	N	N	N
Milestone	N	N	N	N

7.10 SV-9 Systems technology & skills forecast

The SV-9 provides an overview of technological impacts on the operation of the system of systems. These technological advances could be in the form of benign changes such as fuel efficiency increase allowing longer operations or disruptive changes such as added guidance to free-fall bombs. Disruptive changes will ultimately have further impacts on how the system of systems operates whereas the progressive improvement will just improve efficiency of the system of systems. The SV-9 deals with both types as well as impacts due to the change in the skill level of operators (e.g., training impacts). An example SV-9 is given in Figure 105.

The SV-9 is closely connected with the SV-8. The SV-8 is the implementation counterpart to the advances predicted in the SV-9. Both of these views certainly would benefit from modeling but their existence is not useful in selecting a modeling type. In other words, the SV-9 would serve all kinds of modeling types equally well and is not useful in making the decision to pick one modeling type over another. The SV-9 is not a commonly developed product. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a very low creation rate (11%) for SV-9 among the projects analyzed [6]. The results from the discussions below are summarized in Tables 204–207 given in Appendix A.

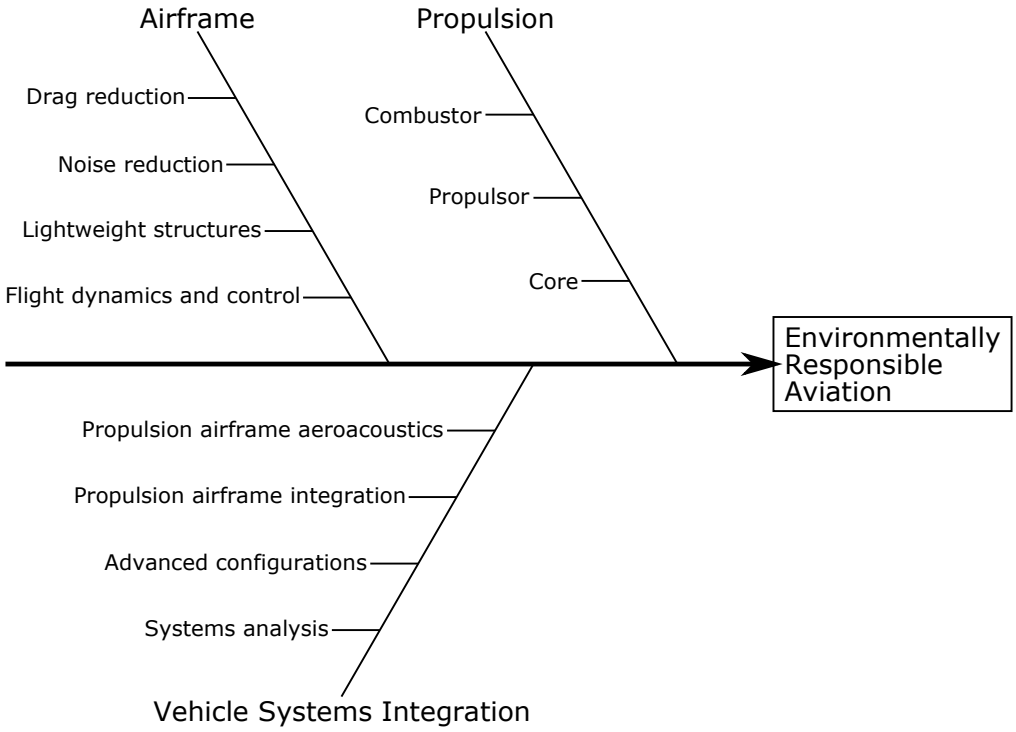


Figure 105: An example SV-9 depicting technology development for commercial aircraft based on NASA's Environmentally Responsible Aircraft program[150].

7.10.1 Graph model

A bipartite graph can be used to represent what improvements are applied to what systems (e.g., higher temperature turbines to fighter aircraft and early warning and control aircraft). While important from a planning point of view, similar to the SV-8, the SV-9 is not a particularly useful source of information for system of systems operations modeling. Table 156 is shows the cautiously optimistic conclusions. Figure 106 shows a plausible case.

Table 156: Mapping between SV-9 and graph model elements

	Vertex	Edge
System	Y	N
Forecast	Y	N

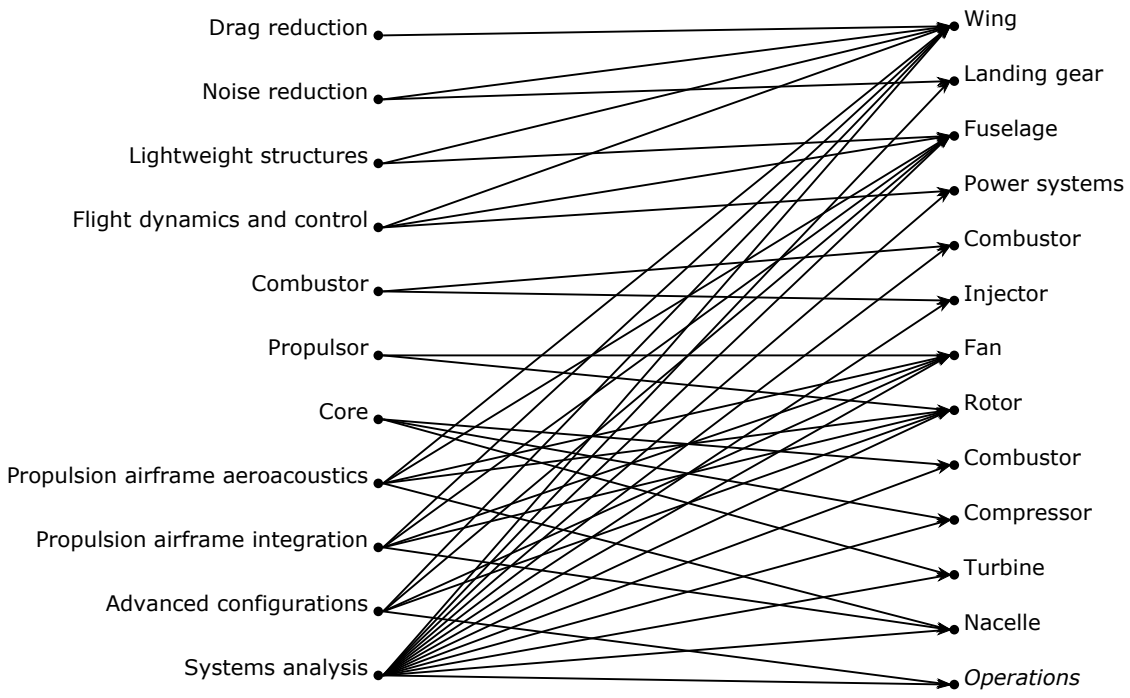


Figure 106: Example SV-9-to-graph model translation based on the SV-9 given in Figure 105. Additional information about what each of the sub-projects deal with are filled in using expert knowledge. This information could be easily included in an SV-9.

7.10.2 Probability model

The SV-9 may include uncertainties in technology forecasts because it is a document about future results; however, these probabilities are not related to the way the system of systems works. Therefore, the SV-9 is deemed to be not suitable for probability modeling. Table 157 shows the results.

Table 157: Mapping between SV-9 and probability model elements

	Conditional probability
System	N
Forecast	N

7.10.3 System dynamics model

The SV-9 does not contain information that can be represented as a quantity that is flowing, transformed, or stored. There is a significant mismatch between what

the SV-9 provides and what a system dynamics model needs. Table 158 shows the negative results.

Table 158: Mapping between SV-9 and system dynamics model elements

	Stock	Flow	Variable
System	N	N	N
Forecast	N	N	N

7.10.4 Markov chain model

The SV-9 does not hold information about a system changing states. Markov chains are not fit to model a system of systems using the information found on an SV-9. Table 159 depicts the negative results.

Table 159: Mapping between SV-9 and Markov chain model elements

	State	Transition
System	N	N
Forecast	N	N

7.10.5 Petri net model

Similar to the discussion above, the SV-9 does not include systems transitioning between states, getting created/removed or merged/split. Therefore, the view is essentially useless for Petri net modeling. Table 160 summarizes the negative results.

Table 160: Mapping between SV-9 and Petri net model elements

	Place	Transition	Arc
System	N	N	N
Forecast	N	N	N

7.10.6 Queueing model

The SV-9 does not have information about job packages queueing up to be processed by a server. The information it provides is not in line with what a queueing model requires. Table 161 shows the negative results.

Table 161: Mapping between SV-9 and queueing model elements

	Arrival	Size	Server
System	N	N	N
Forecast	N	N	N

7.10.7 Discrete event model

Discrete event models are very similar to queueing models in terms of information requirements. Therefore, they too are not suitable modeling methods given an SV-9. Table 162 shows the negative results.

Table 162: Mapping between SV-9 and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
System	N	N	N	N	N	N
Forecast	N	N	N	N	N	N

7.10.8 Agent-based model

The SV-9 includes information about technologies related to the systems themselves or training operators using the systems. Both technologies and operator skills may influence the way systems behave during operations; therefore, it is conceivable that the information given in an SV-9 can be used within an agent-based simulation. However, an SV-9 still lacks significant amount of information that is needed to create an agent-based model. Table 163 reflects the reasoning given here.

Table 163: Mapping between SV-9 and agent-based model elements

	Agent	Environment	Interaction	Rules
System	M	N	N	N
Forecast	N	N	M	M

7.11 SV-10a Systems rules model

The Systems Rule Model sets the rules for system-to-system interaction from a physical perspective. It is similar to the OV-6a, but instead of dealing with operational/business rules, the SV-10a specifies the physical ports, flows, data, and functions. The DoDAF manual states that the SV-10a serves as a definition of implementation logic and identification of resource constraints, both of which are very important in modeling systems and operations. The view goes in detail on under what circumstances the resource and information transfers happen, how system react to external stimuli, the conditions that are required for successful execution of a function (such as proximity during communications), etc. The SV-10a is an invaluable view for all more-granular, higher fidelity modeling types, the kind that goes into detail on how systems and exchanges work rather than them just working with some performance. The SV-10a is not a commonly developed product. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a very low creation rate (11%) for SV-10a among the projects analyzed [6]. The results from the discussions below are summarized in Tables 204–207 given in Appendix A.

> **Example Rule:** If a potential target is not processed or acted upon for more than 15 minutes, assume that it is lost and start the entire process again, i.e., it needs to be searched and found again.

Figure 107: A representative text that can be found in an SV-10a. A similar rule was used in the author's earlier work on a system of systems designed for suppression of enemy air defenses mission [20].

7.11.1 Graph model

The rules as laid out in an SV-10a are not representable using mathematical graphs. This is not unexpected because the same conclusion was reached in the discussion of OV-6a in Section 6.7.1. The rules in an SV-10a are usually stated as sentences but other forms may be possible (such as conditional trees), and the alternative forms may

be represented by graphs. However, the simulation of these graphs will be significantly removed from what the system of systems actually does. Based on this discussion, it can be said that the graph models will not depend on SV-10as and their existence in an architecture will hint that a more detailed, higher fidelity model is more suitable. Table 164 shows the results.

Table 164: Mapping between SV-10a and graph model elements

	Vertex	Edge
Logic & Rules	N	N

7.11.2 Probability model

Similar to the graph discussion, probabilities are not suitable for representing rules that dictate how systems connect to each other and exchange resources. Rules are generally stated in "given A, do B" statements, whereas conditional probabilities are stated as "if A is true, then B will happen with probability X". These statements cannot be translated from one to the other without loss of meaning. Table 165 shows the negative results.

Table 165: Mapping between SV-10a and probability model elements

	Conditional probability
Logic & Rules	N

7.11.3 System dynamics model

The SV-10a does not contain information about quantities flowing between storages, growing or shrinking in numbers by feedback loops. The logic and rule statements of an SV-10a are absolutely incompatible with a system dynamics model formulation. Table 166 shows the negative results.

Table 166: Mapping between SV-10a and system dynamics model elements

	Stock	Flow	Variable
Logic & Rules	N	N	N

7.11.4 Markov chain model

Markov chains model state switches of systems. The SV-10a rules can be thought of as conditions that make systems change the way they are operating (e.g., before condition: idle, after condition: active). While possible for a single system of reasonable complexity, for a system of systems the Markov formulation falls apart. The SV-10a rules are specific to each system or specific to an exchange between two systems. For n systems each with m states the number of total states is m^n, which clearly grows exponentially with n and polynomially with m. Each of these states may be connected with each other and themselves. The number of possible connections grow quadratically, which makes the final Markov chain transition matrix of size $(m^n)^2 = m^{2n}$. It is reasonable to assume that more than 10 rules will be required in the most simple systems and a system of systems will have more than 10 systems, which is hundred billion billion. This is clearly an impractical size for a matrix to work with. Markov chains are deemed to be not suitable for modeling from an SV-10a. Table 167 shows the results.

Table 167: Mapping between SV-10a and Markov chain model elements

	State	Transition
Logic & Rules	N	N

7.11.5 Petri net model

Based on the discussion in Section 6.7.5 Petri nets are well-suited to model rules and state switches associated with them. The discussion is not repeated here for brevity. However, the SV-10a deals with rules that are specific to systems. The main difference is between a Petri net created from an OV-6a and a Petri net created from an SV-10a

is that the SV-10a describes rules about a system or an exchange between systems. The states it describes within the rules is multiplied by the number of systems or exchanges between them. This is not as severe as the Markov chain exponential explosion however. A system modeler will be required to either duplicate places, arcs, and transitions for different systems or add more tokens to the same Petri net and modifying the transition rules where needed. Table 168 shows the results of the discussion for a reasonably simple system of systems. For a highly complex system of systems with many rules and heterogeneous systems the Petri net formulation may turn out to be impractical.

Table 168: Mapping between SV-10a and Petri net model elements

	Place	Transition	Arc
Logic & Rules	Y	Y	Y

7.11.6 Queueing model

Rules states as sentences or even in algorithmic ways are not suitable for queueing models. The reader can see how the contents of an SV-10a are conceptually different than the definition of a queueing model. A statements such as "if System A receives Order B, it performs Function C" simply cannot be turned into a job that is queueing up to be performed by a server. The information types provided by the SV-10a and required by queueing models are mismatched absolutely. Therefore, Table 169 shows completely negative results.

Table 169: Mapping between SV-10a and queueing model elements

	Arrival	Size	Server
Logic & Rules	N	N	N

7.11.7 Discrete event model

Discrete event models are conceptually very similar to queueing models; and therefore, the mismatch between the architecture view and modeling remains largely unchanged.

There are a few differences however. Discrete event models do allow for some logic to be included within the network of job stations. Some rules specified in an SV-10a may dictate how the job queues behave, e.g., first in first out, last in first out, or priority. They may also dictate how to deal with jobs that fail, e.g., discard, retry, or try something else. Figure 108 depicts a decay logic. Based on this discussion, it is decided that an SV-10a may be containing information to fill in the details of a discrete event model but is not suitable to build its structure. Table 170 summarizes the results.

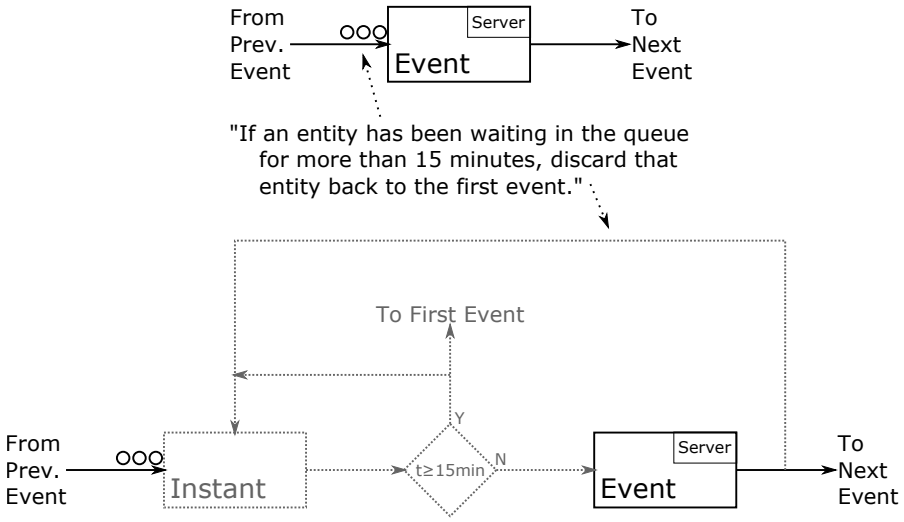


Figure 108: Example modifications to a discrete event model based on an SV-10a rule

Table 170: Mapping between SV-10a and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
Logic & Rules	N	M	N	N	N	N

7.11.8 Agent-based model

The SV-10a includes a significant amount of information about how systems work internally as well as interact with each other. The view states rules that dictate how systems behave and that is the main piece of information a system modeler needs to

build an agent-based model. The rule statements will provide a system modeler a list of agents expected to be in the model, the way they interact with the environment (e.g., can float, can fly, can move, can climb), as well as detailed information on how the systems interact with each other using resource ports and the rules associated with each of the resource exchanges. The inclusion of an SV-10a is a good indication that the system of systems requires the kind of simulation that includes direct and detailed analysis of the consequences of the rules listed in the SV-10a. Only an agent-based model has such a high fidelity. Table 171 lists the results of the discussion.

Table 171: Mapping between SV-10a and agent-based model elements

	Agent	Environment	Interaction	Rules
Logic & Rules	M	M	Y	Y

7.12 SV-10b Systems state transition description

The Systems State Transition Diagram depicts the order behind the systems changing their states due to external or internal reasons such as functions, receipt of information, or interaction with another system. A practical example could be a sensor platform transitioning from detection mode to tracking mode once a target is found. This arrangement is of course the same as the OV-6b Operational State Transition Diagram. Because the SV-10b is dealing with discrete states and events, a discrete event model is probably the best-suited modeling type to represent it in a dynamic simulation. Most elements needed for a discrete event model are present in the SV-10b (queue logic and resource use may be missing).

The SV-10b is also similar to the SV-4 but with more detail. The SV-10b shows the state transitions that enable the workflow depicted on the SV-4. An example SV-10b is given in Figure 109. In many cases an SV-4 can be reduced from the SV-10b; however, during an early phase of architecture design, the opposite direction is taken. The designer then needs to check for inconsistencies after the fact. Architecture design

software packages will warn users for inconsistencies or even partially auto-populate the SV-10b based on the SV-4 the user has created earlier [CITE IBM]. Even when not creating discrete event models, the SV-10b is an essential view for modeling. It is the view that shows how the system really work on the inside and can be used to specify the internal processes for agent-based models just as well. The SV-10b is surprisingly not a commonly developed product. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a very low creation rate (8%) for SV-10b among the projects analyzed [6]. The results from the discussions below are summarized in Tables 204–207 given in Appendix A.

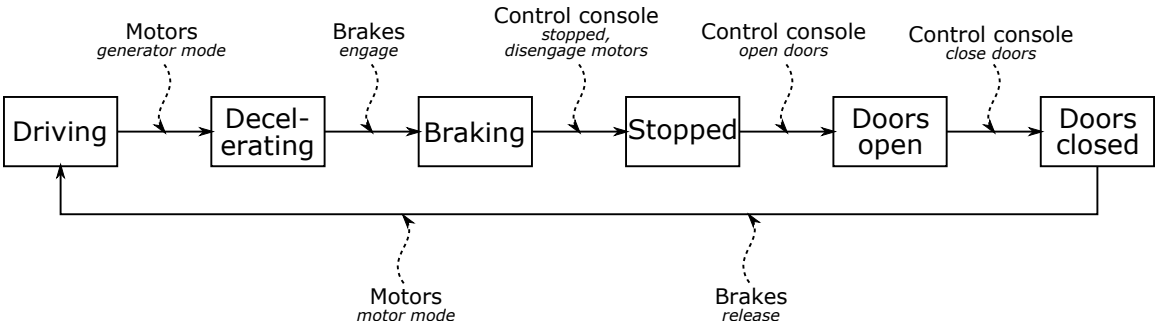


Figure 109: An example SV-10b depicting the operation of a train. It shows more detail about the train system compared with the SV-4 given in Figure 94.

7.12.1 Graph model

A graph model of manageable complexity can be constructed from an SV-10b for a single system transitioning between its operation states. However, scaling this model to the entire system of systems is highly unlikely to succeed. The states of the entire system of systems grow in number geometrically with respect to number of systems (i.e., exponentially, see Section 7.11.4 for discussion). Even smaller-scale systems of systems easily over-extend the capabilities of human modelers. Algorithmically, a graph model can be constructed from a combination of SV-10bs for different systems and a number of systems but the resulting model will be neither human-readable nor highly useful. At this level of detail, the modelers should think about higher

fidelity modeling techniques that use less abstraction and require the type of detailed information the SV-10b is already supplying. Table 172 shows positive results that should be interpreted with the caveats discussed here.

Table 172: Mapping between SV-10b and graph model elements

	Vertex	Edge
System state	Y	N
Function	N	Y

7.12.2 Probability model

For each of the transitions the SV-10b a probability of success can be assigned to them. This probability is in the form of a conditional probability, because it does not take into account whether other functions have failed or not. The probability value may be included on the SV-10b but it is not a standard practice. Instead, the probability value could be included within the SV-7 matrix and be linked to the function that makes systems to switch their states. Table 173 shows the results. The probability model suffers from the same state space explosion as the graph model discussed above.

Table 173: Mapping between SV-10b and probability model elements

	Conditional probability
System state	N
Function	Y

7.12.3 System dynamics model

The SV-10b shows systems switching between their states, which is a discrete step. Such discrete state transitions are extremely difficult to model in the system dynamics paradigm which solves a system of differential equations. These equations must be differentiable everywhere and discrete switches are neither differentiable nor continuous. Some system dynamics simulation software use smoothing or other advanced

methods to allow discrete switches in the models; however, these either approximate the discrete nature of the systems or run into convergence problems. The description the SV-10b provides has a mismatch with the types of systems a system dynamics model is trying to represent. The argument in the OV-6b—the operational view counterpart of the SV-10b, see Section 6.8.3—discussion seems to be in conflict with the argument provided here initially. However, the reader must remember that the SV-10b is highly specific for a single (or a single type of) system, whereas the OV-6b takes a bigger picture perspective. Therefore, if there are a large number of systems transitioning between operational states, the many discrete changes can be validly modeled as a smooth transition (i.e., a stair looks like a slope if seen from afar). Table 174 summarizes the results. While conceptually unfit, some SV-10bs can be modeled using the system dynamics paradigm.

Table 174: Mapping between SV-10b and system dynamics model elements

	Stock	Flow	Variable
System state	M	N	N
Function	N	M	M

7.12.4 Markov chain model

Markov chains are well suited for representing state changes on a system and are frequently used to study engineering designs [92] and even biological systems [82]. As such, a Markov chain is suitable to simulate an SV-10b; the mapping is immediate: system states map to Markovian states, function map to Markovian transitions. However, the reader is reminded that with the number of systems growing and their states becoming more complicated, the number of states for the entire system of systems becomes impractical. Because the ultimate goal is to create models that simulate the behavior of the system of systems not its individual architecture views, it is concluded that the inclusion of an SV-10b gives no indication whether a Markov chain is a good fit for representing the system of systems. Table 175 shows the results.

Table 175: Mapping between SV-10b and Markov chain model elements

	State	Transition
System state	M	N
Function	N	M

7.12.5 Petri net model

Petri nets are much more suitable for state switching in situations where multiple but similar systems are operating in unison. Heterogeneous systems can be added by using colored Petri nets and minimal addition of new states. Being able to track multiple tokens (i.e., systems) in the same state network gives Petri nets an edge over Markov chains by avoiding the majority of the geometric explosion of states/places. However, their execution is significantly slower than Markov chains. The system states can naturally map to Petri net places while the functions that transition systems between states can be represented as arcs and transitions. Table 176 shows the results.

Table 176: Mapping between SV-10b and Petri net model elements

	Place	Transition	Arc
System state	Y	N	N
Function	N	Y	Y

7.12.6 Queueing model

The queueing model has some overlap with the SV-10b. The view is depicting systems changing their states based on some internal or external influence. This can be easily recast into a concept that takes systems and moves them from state to state based on a work done by a server. Such a view is also compatible with a large number of systems. However, the state transitions in the SV-10b are based on conditions not job durations. Unfortunately, this is not a reconcilable difference; the two approaches differ at a fundamental level. Based on this significant difference, queueing models are deemed unfit for modeling systems of systems that require specific SV-10b definitions. Table 177 shows the results in tabular form.

Table 177: Mapping between SV-10b and queueing model elements

	Arrival	Size	Server
System state	N	N	N
Function	N	N	N

7.12.7 Discrete event model

The SV-10b and SV-10c are both very useful views for discrete event modeling. The same argument as above can be made about timing information being missing from the SV-10b for properly set up a discrete event model; however, that information can be read on an SV-10c the two views can complement each other very well. Discrete event models can represent routing based on reasonably complex logic (e.g., check for expendable resources and rerouting based on conditions and history) and track properties of entities tracing a path through states. These two additional modeling freedoms discrete event models enjoy over queueing models make the content of an SV-10b attractive for modeling. The functions can be modeled as events that transition systems modeled as entities from state to other states. Some systems will serve as servers that perform the functions as well. Table 178 shows the results of the discussion and Figure 110 shows an example.

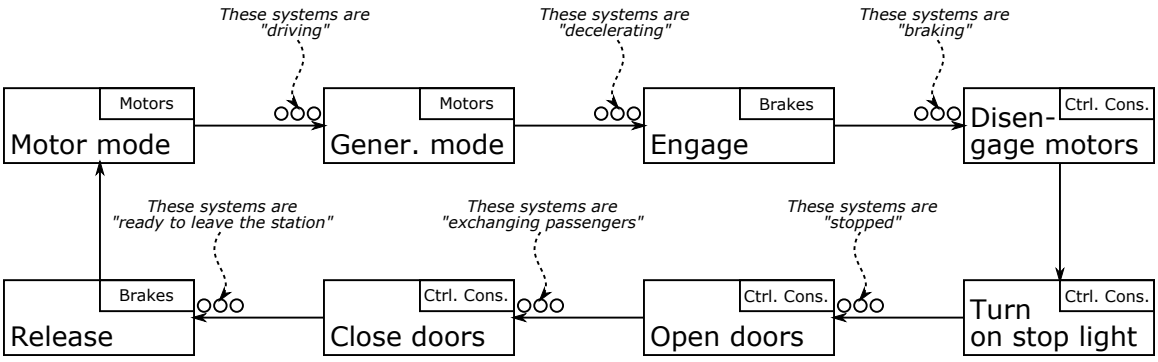


Figure 110: Example SV-10b-to-discrete event model translation based on the SV-10b given in Figure 109.

Table 178: Mapping between SV-10b and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
System state	N	M	N	N	Y	N
Function	Y	N	Y	M	N	M

7.12.8 Agent-based model

The behaviors of agents representing systems can be modeled using SV-10b. The SV-10b shows how the systems react to internal and external functions by changing their states of operation. These discrete state changes ultimately dictate what the system will be doing next. An SV-10b is a perfect fit for modeling the internal logic of an agent in an agent-based model. Figure 111 shows an example. Since the states map to the internal logic of an agent, the systems are represented as agents. There is a lack of environment definition in an SV-10b, which will most likely be supplied by operational views. Additionally, the mapping is not one-to-one, i.e., the functions on the view provide information about interactions between agents but also include information about agents and their rules. The translation from the architecture view to the model requires interpretation. As with all views, the SV-10b is meant to be read by humans, not by computers that can automatically transform it into executable simulation models. Table 179 summarizes the discussion in tabular form.

Table 179: Mapping between SV-10b and agent-based model elements

	Agent	Environment	Interaction	Rules
System state	Y	N	M	Y
Function	Y	N	Y	Y

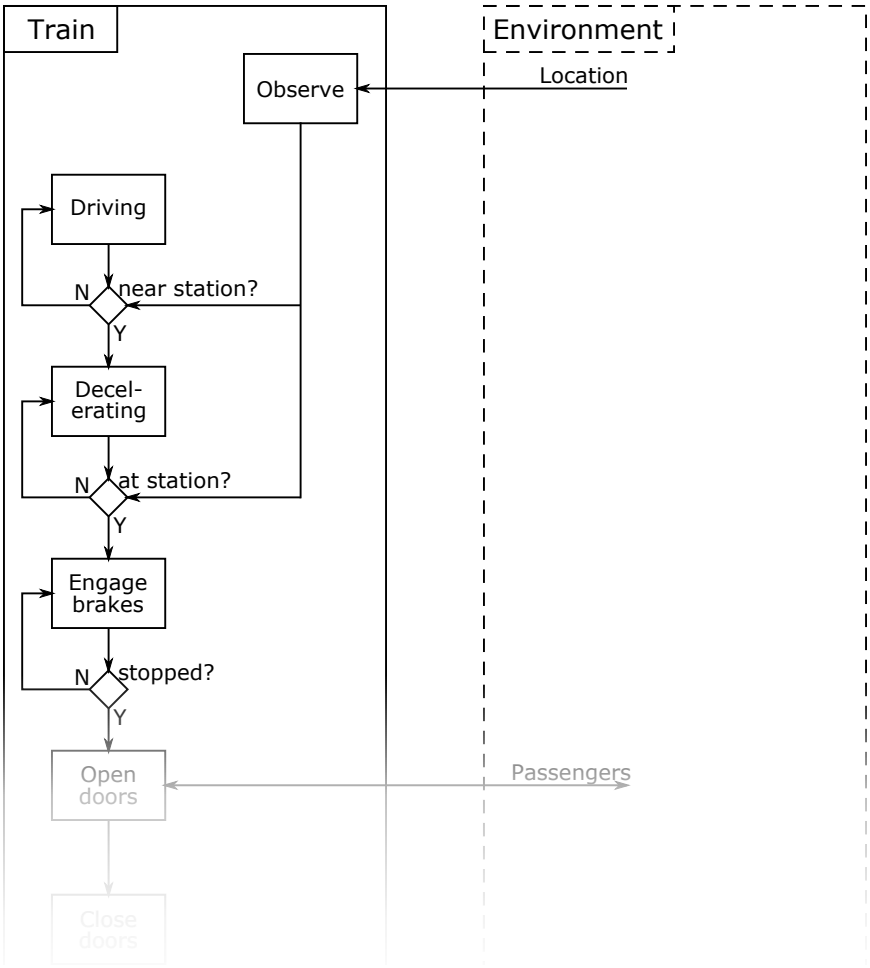


Figure 111: Example SV-10b-to-agent-based model translation based on the SV-10b given in Figure 109. Notice that the environment is represented with dashed lines because it is missing in the SV-10b.

7.13 SV-10c Systems event-trace description

The Systems Event-Trace Description is the OV-6c counterpart for the systems views. Instead of operation taking time on lifelines, it depicts systems/performers being busy and the resource flows they create and consume. The view must be limited in scope because including every system and performer as a lifeline would not be practical. The SV-10b is therefore, always created with a scenario in mind and the DoDAF manual explicitly states that a scenario description must accompany an SV-10c[59]. An example is provided in Figure 112.

The main idea the SV-10c shows is the dependence of systems on other systems'

products or effects, i.e., the cases when a system needs to wait for another system to perform a task before the original system can perform its task. Additionally, it introduces the timing element to the system views: duration, simultaneity, and concurrency. Just like the SV-6c, timing information is crucial for discrete event and agent-based modeling efforts. The SV-10c is also surprisingly not a commonly developed product. DoDAF Product Development Questionnaire Analysis Report and New Product Recommendations Report cites a very low creation rate (24%) for SV-10c among the projects analyzed [6]. The results from the discussions below are summarized in Tables 204–207 given in Appendix A.

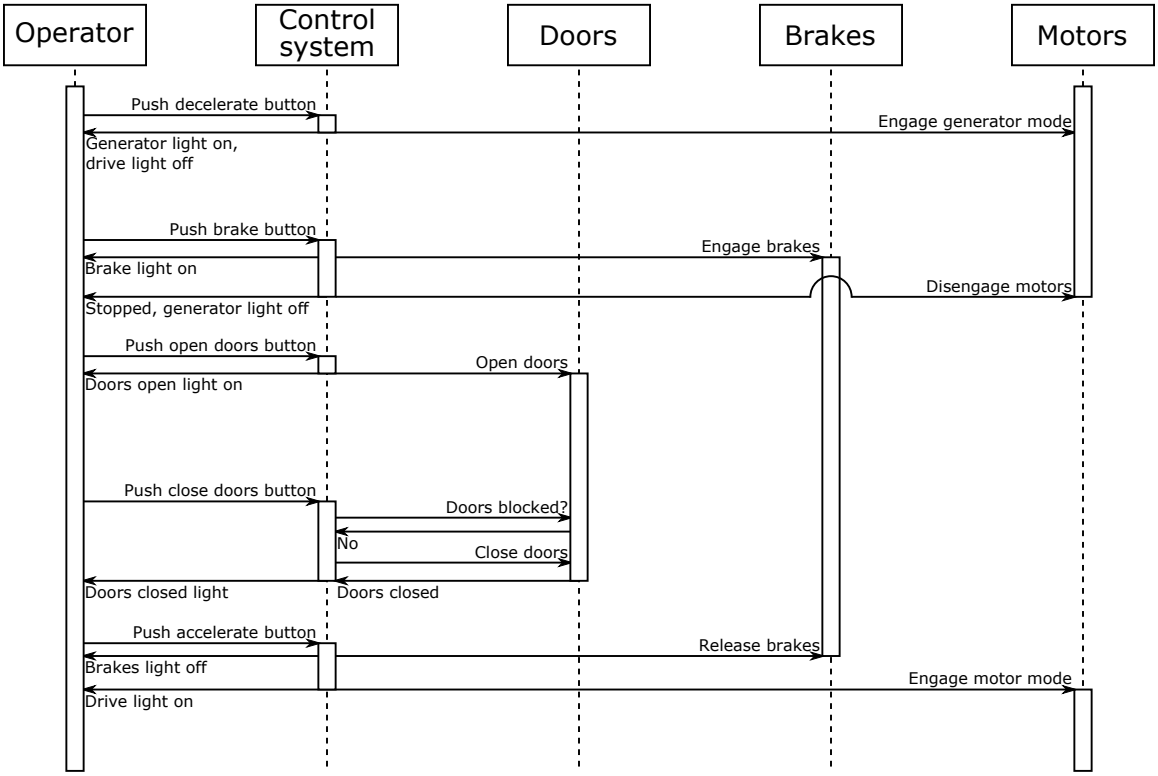


Figure 112: An example SV-10c depicting the operation of a train. It shows significantly more detail compared with the SV-4 given in Figure 94.

7.13.1 Graph model

The SV-10c can be recast as a mathematical model because each activity is connected to other activities by interactions and condition interactions only connect to activities;

these are the exact rules that govern graph models: vertices are connected to other vertices by edges and each edge connects exactly two vertices. Therefore, the SV-10c can be represented and simulated using a graph model. The problem that arises is that the timing information is completely lost. The most important information on an SV-10c is the timing information and the reader is reminded that the goal of modeling is not the mere translation of what is on a view to a working model; it is to model the system of systems the views are describing. As such, creating a graph model from an SV-10c will not provide the system modeler any useful information about the system of systems. The connectivity of activities and the shortest paths and maximum flows between them are essentially meaningless metrics. Table 180 is cautious in specifying entries as "maybe"s instead of "no"s. It is conceivable that a graph model out of an SV-10c could be useful for a specific purpose not considered here.

Table 180: Mapping between SV-10c and graph model elements

	Vertex	Edge
System	M	N
Functions	M	N
Interactions	N	M

7.13.2 Probability model

Conditional probabilities may be used to model success rates for functions or "interactions"; however the focus of an SV-10c is timing of functions not their likelihoods of success. The information needed to create a probability model is missing from the SV-10c and the goals of the model and the view do not match. Therefore, it is unlikely that a probability model representing a system of systems to be constructed out of that systems of systems' SV-10c. Table 181 shows the results.

Table 181: Mapping between SV-10c and probability model elements

	Conditional probability
System	N
Functions	N
Interactions	N

7.13.3 System dynamics model

The SV-10c depicts systems performing a number of functions that create a desirable effect. The functions can be immediate or have finite durations and after they are finished, their products are passed along to other systems so they can perform their part. In a sense, there is a flow products between system functions; however, the transfers are unit transfers, and there is no continuous kind of flow that can be modeled using a system dynamics model. A Markov chain or Petri net would be better suited for such discrete transitions between functions. Table 182 summarizes the negative results.

Table 182: Mapping between SV-10c and system dynamics model elements

	Stock	Flow	Variable
System	N	N	N
Functions	N	N	N
Interactions	N	M	N

7.13.4 Markov chain model

Markov chains are somewhat suitable to model the process depicted in an SV-10c. The transitions between Markov states can be used to represent the hand-offs between functions. If the hand-off is instantaneous, the architecture to model mapping is trivial but if the hand-off is not instantaneous, an artificial state can be added to the Markov chain model to simulate the duration of the transfer. The modeler can also opt to use systems (lower fidelity) or the functions they perform (higher fidelity) as states. However, the definition of states is not straightforward. At each instant in time, more than one function can be ongoing (concurrency can be represented easily

in an SV-10c); however, this is not possible with Markov states. For each possibly overlapping functions, multiple states must be created (see Figure 113). The same process must be performed whether the system modeler chooses to model systems as states or functions. Overall, it is possible to use Markov chains for modeling specific operations of a system of systems but the model creation is not straightforward. Table 183 summarizes the results.

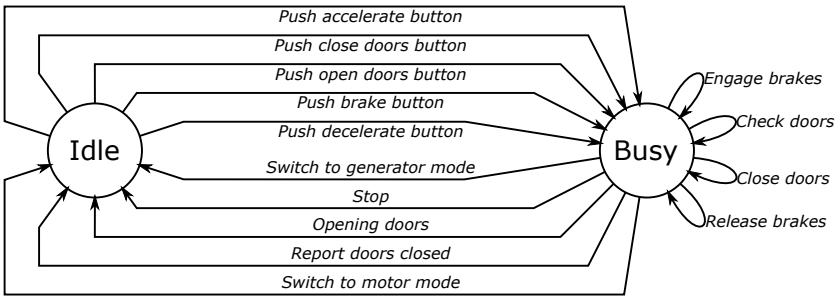


Figure 113: Example SV-10c-to-Markov chain model translation based on the SV-10c given in Figure 112. This model only depicts the control system in isolation from other subsystems, which is not realistic and can only be used with limited success. If other subsystems are to be included in the model, the overlapping states must be split into numerous states as only one Markov state can be active at any given time as discussed in the text.

Table 183: Mapping between SV-10c and Markov chain model elements

	State	Transition
System	Y[1]	N
Functions	Y[1]	N
Interactions	N	Y

[1] Systems and functions do not map to states simply due to the possible concurrency of functions depicted in SV-10cs.

7.13.5 Petri net model

A Petri net is highly suitable for translating an SV-10c to a simulation model. Each function can be represented as a place within a Petri net and multiple places can be active by using multiple tokens. These tokens will travel from place to place (i.e., from

function to function) via the arcs and transitions that represent the interactions. In the Petri net formulation a transition can merge or split a token, which is also allowed in an SV-10c. A Petri net can also capture the multiple requirements of a function before it can execute. In conclusion, a Petri net can handle the many flexibilities offered by an SV-10c. Table 184 shows the positive results.

Supporting the argument above, the literature has many examples that deal with translating SV-10cs to Petri nets. Wagenhals et al. use a number of SV-10c-like diagrams to construct colored Petri nets and even perform the translation algorithmically [190]. They do not use DoDAF but their UML diagrams are similar to SV-10b/cs. Their findings agree with the arguments set forth in this thesis: Petri nets are a viable option for modeling from SV-10cs and the existence of an SV-10c can be used as a hint that the system of systems in question can be successfully modeled with Petri nets. The details about what elements gets translated into what modeling element are different in their work due to the different architecture languages used (UML vs. DoDAF).

Wang and Dagli use two SysML diagrams to create colored Petri nets using a schema [192]. The diagrams used are internal block diagrams and sequence diagrams. The internal block diagram which can be compared to a detailed SV-1, SV-2, and SV-5a combination is used to layer the modeling effort; start with smaller models within relatively independent activities and gradually moving towards the entire operation. The sequence diagrams, which are very similar to SV-10cs, are then translated into colored Petri nets to model the these smaller unit models. They are then put together to create the model for the entire operations. Therefore, it can be argued that Wang and Dagli's work agrees with he arguments set forth in this thesis: SV-10cs are highly suitable for Petri net modeling.

Bai et al. use Petri nets to validate architectures specified using UML through simulation [199]. In their process a number DoDAF views are prepared including

an SV-10c as well as an SV-6[4].They then model the systems using UML sequence diagrams that are translated into Petri nets automatically. Their success highlights how natural it is to model systems of systems with Petri nets given their SV-10c views. with some accompanying information such as resource transfers and metrics. Bai et al.'s results are in support of the position taken in this thesis.

Table 184: Mapping between SV-10c and Petri net model elements

	Place	Transition	Arc
System	N	N	N
Functions	Y	N	N
Interactions	N	Y	Y

7.13.6 Queueing model

Queueing model is also a viable option for building a model from an SV-10c. The functions can be represented as stations within a queueing model (i.e., size of a job because they include information on how long the function actually takes to complete). The systems that perform the functions can be represented as the servers doing the job, and finally the hand-offs between the functions can me modeled as departures from job stations and arrivals to other job stations. A queueing network can then be constructed with minimal loss of fidelity (some logic operations may not be possible). Table 185 summarizes the results.

Table 185: Mapping between SV-10c and queueing model elements

	Arrival	Size	Server
System	N	N	Y
Functions	N	Y	N
Interactions	Y	N	N

[4]The authors refer to an SV-3 in their process but this appears to be a typographical error. They actually usE an SV-6. This is evident by the name fo the matrix in the text and their UML-to-DoDAF translation figure.

7.13.7 Discrete event model

Discrete event is arguably the best choice to model from the information included in an SV-10c. It enables representation of functions taking finite time, conditions they may require to be met before executing, systems acting like servers that perform the functions and able to be set as idle or busy, various failure and success scenarios, as well as the ability to add queues, and spending of resource that may be implicit but rarely explicit in an SV-10c. The mapping from architecture elements to modeling elements follows the same form as the queueing model. Here, functions are modeled as events that occur after some set time of a process. The queues are defined for each function. The entities are transported between processes by transition (i.e., interactions), and systems perform the processes that fire the events. While queueing model seems to be a simpler model to build from an SV-10c, it must be said that many of the requirements for analytic solutions are likely to be not met with practical examples of an SV-10c; therefore, it will revert to be simulated just like a discrete event model anyway. Table 186 shows the results in a tabular form.

Table 186: Mapping between SV-10c and discrete event model elements

	Event	Queue	Transition	Server	Entity	Resource
System	N	N	N	Y	Y	N
Functions	Y	M	N	N	N	M
Interactions	N	N	Y	N	M	N

7.13.8 Agent-based model

The SV-10c is also a valuable architecture view for agent-based modeling. Within it, agents can be identified along with their internal processes. Each of these processes, are coupled with triggers and effects. A trigger such as a receipt of a message can make an agent to perform that specific process which can create another message for another agent to receive and trigger one of its own internal processes. Figure 114 shows an example transformation. The timings and requirements of each of the

functions will provide a modeler with many of the rules—other rules will be found in the SV-10a—needed to create a realistic agent-based model. An SV-10c can also include the operating environment as another timeline, mimicking a system. Table 187 summarizes the positive results.

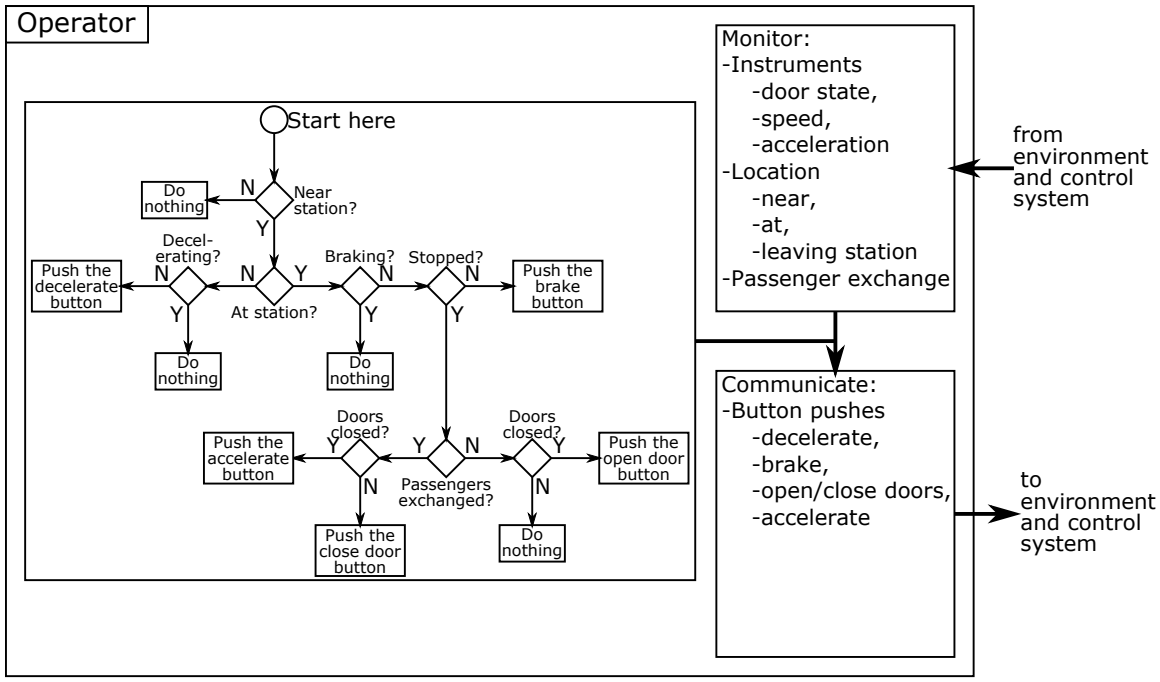


Figure 114: Example SV-10c-to-agent-based model translation based on the SV-10c given in Figure 112. This model only depicts the internal logic for the operator agent within the train system. The agent communicates with the environment and control system only because the SV-10c does not show any subsystems interacting with the operator.

Table 187: Mapping between SV-10c and agent-based model elements

	Agent	Environment	Interaction	Rules
System	Y	M	N	N
Functions	N	N	N	Y
Interactions	N	N	Y	Y

CHAPTER VIII

EXPERIMENTAL TESTING OF THE ELEMENT MAPS

> Matematik esas olarak sabır olayıdır. Belleyerek değil keşfederek anlamak gerekir[1].
>
> Cahit Arf

This chapter details the testing of the tables compiled in Chapters 6 and 7. As discussed in Section 3.3.1 the tests are carried out using the four examples.

8.1 Experimental setup

Each of these cases will now be modeled using conceptual models that can be readily simulated with a simulation engine. These conceptual models will then be checked against the architectural models and matches and mismatches will be highlighted. With these experiments the equivalence of architecture models and conceptual models will be tested.

Before the experiments are discussed it will be worthwhile to remember the distinction used in this work between a conceptual model and simpler descriptive models. A conceptual model is a specific kind of descriptive model whose translation to a simulation is readily known, i.e., it conforms to a description standard that is known to have an algorithmic solution. With this definition, the reader can see that not every physical or operational description will fit the accepted norms of various modeling techniques or be complete enough so that the simulation can run without getting stuck. Both fitting the norm and completeness is important for a descriptive model to be a conceptual model.

[1]Author's translation: Mathematics is essentially a matter of patience. We must gain understanding by discovery, not by committing to memory.

With these experiments the groundwork is done for the straightforward translation of architectural models to conceptual models. Therefore, while the architectural models may not fit within the traditionally accepted norms for various modeling techniques, examples will be given to show that they can be converted into such forms and are, therefore, equivalent to them. The process is depicted in Figure 115. The chapter contains experiments of three types:

Type A Experiment Used for cases when a modeling type is predicted to be *impossible* by the tables given in Chapters 6 and 7. After listing hypothetically impossible modeling types, for each do:

- Attempt building a model starting from the most useful architecture view
- Comment on what is missing
- Try filling in the information by adding other provided views until information runs out or model is complete
- Comment on whether the build was a success or not based on whether a part of model structure is still missing

Type B Experiment Used for cases when a modeling type is predicted to be *possible but not necessarily so* by the tables given in Chapters 6 and 7. After listing hypothetically possible but not necessarily so modeling types,

- Attempt building a model starting from the most useful architecture view (call this Model X)
- Attempt building a similar[2] model that is predicted to be definitely possible from the same architecture view (call this Model Y)
- Compare the two models

[2]Similar in the information it uses and optionally the way it works

  - How is Model X less structured than Model Y?
  - What causes the Model X to require more information than Model Y?
  - Can these assessments be generalized?
- Comment on what is missing from Model X
- Try filling in the information by adding other provided views until information runs out or model is complete
- Comment on whether the build was a success or not based on whether a part of model structure is still missing

Type C Experiment Used for cases when a modeling type is predicted to be *definitely possible* by the tables given in Chapters 6 and 7. After listing hypothetically definitely possible modeling types,

- Attempt building a model starting from the most useful architecture view
- Comment on what is missing
- Try filling in the information by adding other provided views until information runs out or model is complete
- Comment on whether the build was a success or not based on whether a part of model structure is still missing

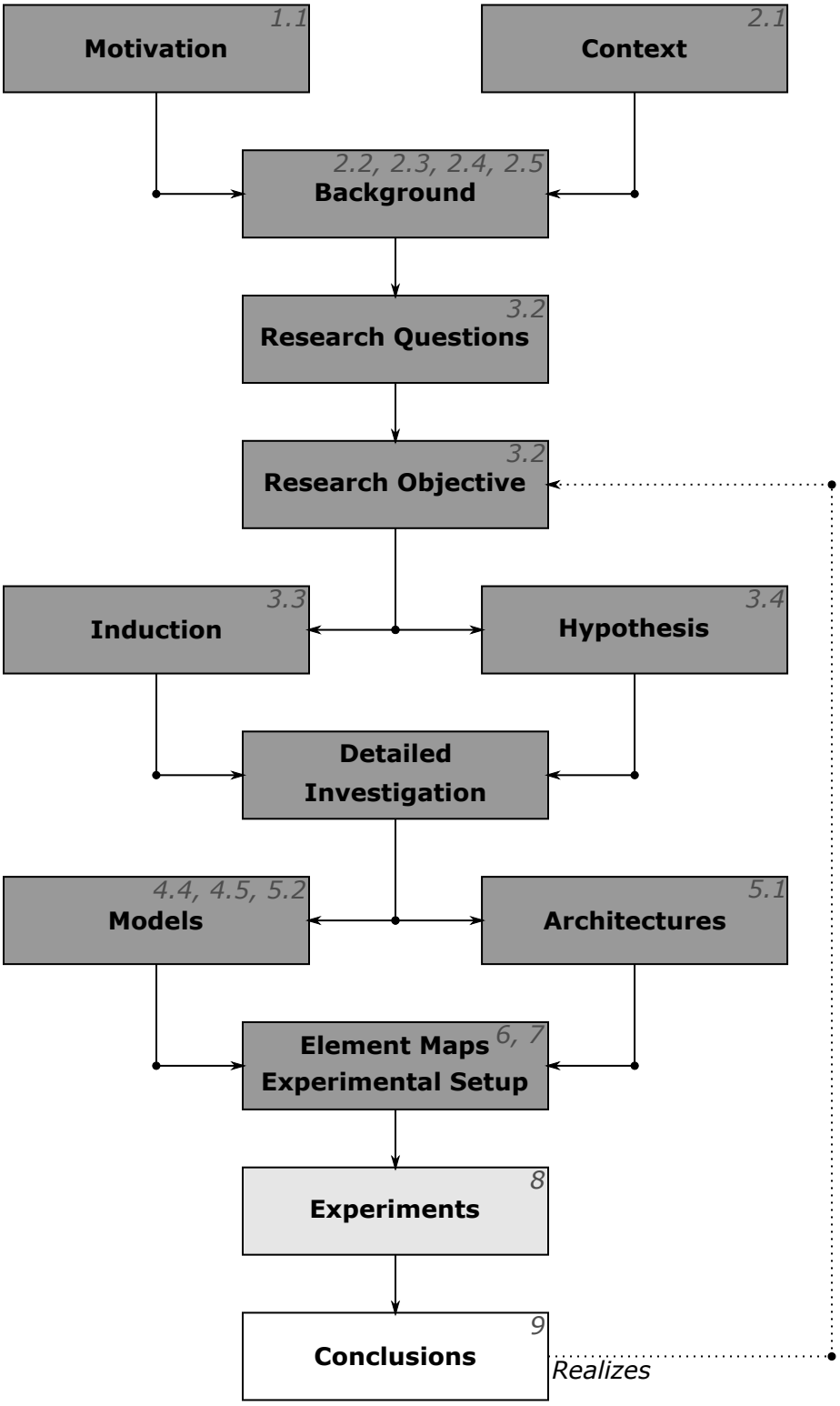


Figure 115: This chapter tests the element maps created in the last two chapters.

8.2 2012–2013 Real World Design Challenge State Aviation Problem

The cases are organized from the smallest to the largest in system of systems size to ramp up the difficulty smoothly. The first case to be investigated is the 2012–2013 RWDC State Aviation Problem [51]. The challenge scenario is as follows.

> "Search for a missing, injured and immobilized child with a blue jacket during the day at the Philmont Ranch in a designated 2-mile radius circular search area. This area is sized to allow line-of-sight contact between the operator and aircraft to be maintained, per FAA guidelines. Teams should refine the vehicle design, sensor payload selection, search pattern, best altitudes for the selected sensor payload, and associated ground equipment to find the child in the minimum time while also minimizing cost."

This problem interfaces several systems together but its scope and complexity are severely limited to a small range of use cases. Some of the architecture models created and used for this problem have been introduced in the earlier chapters as simple examples to various system views and they will be used here again to explain how architecture elements can be procedurally turned into executable modeling elements.

Jones Wyatt uses an SV-1 (Figure 116), an SV-6 (Table 129), and an SV-7 (Table 188) as shown in Table 189. Some the views she uses come from the problem definition (such as the SV-1 and SV-7) but the SV-6 was specifically constructed to perform the interoperability analysis. Using the connections and the resources they carry between systems, Jones Wyatt performs reliability calculations on the entire connected graph model she derives from the architecture models [111].

To analyze interoperability between systems, Jones Wyatt naturally uses system viewpoints. Also, she strategically uses views that do not describe order of execution, a scenario, a timeline, or further functional details. Her choice simplifies modeling appropriately for interoperability studies. Interestingly, based on the previous chapters,

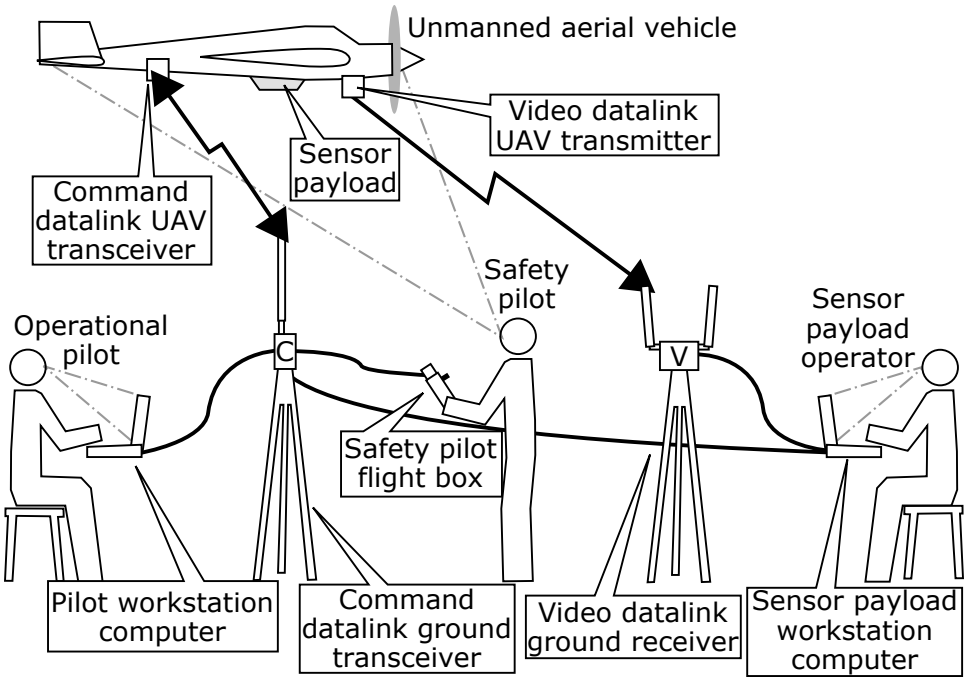


Figure 116: Jones Wyatt's SV-1[111] for the RWDC problem (Adapted from [51])

her choice of architecture views appears to contain enough information to support higher fidelity modeling such as discrete event or agent-based modeling. The major observation extracted from her work is that the architectural model elements map to conceptual model elements fairly well for all modeling types other than agent-based modeling, as can be seen in Table 190.

Table 188: Jones Wyatt's partial SV-7 based on her UAV design from the options provided by the RWDC problem (reproduced from her thesis[111])

Component	Description	Power usage (Watts)	Required quantity	Per item cost (USD)
Video datalink UAV transmitter	The device which transmits the video captured by the payload to the ground control system.	0.4	1 for each sensor payload	200
Command datalink UAV transceiver	The device which sends and receives communication signals between the FCS and the Pilot Workstation.	0.3	1 for each UAV	300
Flight control system	The system which actually controls the aircraft and communicates with the pilot in the ground station. Functionality includes GPS navigation and telemetry, ability to relay sensor payload commands, ability to implement repetitive sensor payload routines (e.g. sweeping pan back and forth), and semi-autonomous waypoint following capabilities.	0.1	1 for each UAV	2,000
Battery	Light-weight batteries with enough energy to supply various UAV components. COTS solution.	n/a	1 for each UAV	Market price
Sensor payload	One of the provided Daylight Electro-Optical Camera options; the mid-level sensor was chosen.	10	1 (for this scenario)	38,000

Table 189: The viewpoints developed for Jones Wyatt's RWDC architecture.

Viewpoint	Jones Wyatt RWDC
OV-1	
OV-2	
OV-3	
OV-4	
OV-5a	
OV-5b	
OV-6a	
OV-6b	
OV-6c	
SV-1	✓
SV-2	
SV-3	
SV-4	
SV-5a	
SV-5b	
SV-6	✓
SV-7	✓
SV-8	
SV-9	
SV-10a	
SV-10b	
SV-10c	

Table 190: Observations from the Jones Wyatt's RWDC architecture

Model	Feasible	Views ordered by utility
Graph	✓	SV-1=SV-6
Probability	✓	SV-6
System dyn.	✓	SV-6, SV-1
Markov chain	✓	SV-6
Petri net	✓	SV-6
Queueing	✓	SV-6
Disc. event	✓[1]	SV-6, SV-1
Agent-based	×	SV-1=SV-6

[1] Many Discrete Event elements share the same architecture elements. Depending on how detailed the architecture models are, this conversion may be impossible. It will be tested.

8.2.1 Creating models from RWDC architecture views

The first experiment to be attempted is the agent-based model. It appears to be impossible as the information about the environment seems to be completely missing. This guess will be tested in the first experiment below. Second experiment will be a discrete event modeling attempt. There are architecture elements that map to multiple modeling elements. If these architecture elements do not include significant amounts of information, it may be impossible to define modeling elements from them in an executable way. This will be investigated in the second model. The third experiment to be tried is a system dynamics model. The system is mostly based on a communications model and it is interesting to see if generation and dissemination of data can be modeled using a system dynamics approach. The architecture views appear to be very useful in the creation of this type of model. Finally, Jones Wyatt's own probability model will be discussed. Her own work will not be used as an example for this thesis, because she created the views and models herself, which can introduce bias to result. It is however a good demonstration of how architectural elements can be mapped to various modeling elements.

8.2.1.1 Building an agent-based model (Type A experiment)

Investigating the architecture views included in Jones Wyatt's work, Table 190 suggests that the an agent-based model is not viable. Type A experiments were defined to attack the hypothesis and prove it to be wrong by attempting to build a model predicted to be impossible. An agent-based model construction is attempted here. Table 190 suggests that the SV-1 and SV-6 views are equally viable as a starting place. The views are given in Figures 116 and Table 129 respectively.

The first step in the modeling effort is to define what will be modeled as agents and to determine their functions. In this example, a Python script is used because the object-oriented coding is especially practical for agent-based modeling. Based on the

two views used, the two agents defined in the model are "Systems" and "Interfaces". Resources (data items) could have been modeled as agents as well; however, there is no description about them and defining a class based on no information was deemed to be counter-productive. Attributes and methods for the two agent classes are given below.

System Attributes model The model this system is part of (this is needed for mechanistic reasons)

name A name for the agent, which must be unique

interfaces An associative array that determines which data type gets pushed on which interface

data generated If the agent is continuously generating data, it must be declared here

data manipulations Agents receive data, act on them, and send results back. This associative array holds recipes for what happens to each type of received data.

incoming data A *set* that will change at each step

outgoing data A *set* that will change at each step

Methods add interface to Interfaces are added to the system using this method. The sink system and data type must be specified.

add data manipulation Recipes are added to the data manipulations attribute

send data When the system is ready, it will load data to appropriate interfaces

receive data When the system receives data, it adds the data to the incoming data set

act on data Data in manipulated and added to the outgoing data set

generate data If the system generates data on its own, it will happen using this method

step Every agent must have a step method. This special method will call on other methods so that the agent behaves correctly.

Interface Attributes model The model this interface is part of (this is needed for mechanistic reasons)

data description The type of data the interface carries

source The source of the data

sink The recipient of the data

loaded data Currently loaded data on the interface

Methods load data Using this method, the source system loads data to the interface

carry data When its ready, the interface will carry the loaded data to the sink system

step Every agent must have a step method. This special method will call on other methods so that the agent behaves correctly.

The model includes the systems and the interfaces in its own attributes (as lists or associative arrays). It also deals with the execution of the whole simulation. It is not interesting to report on the mechanics on how the model class works. Finally, systems and interfaces are instantiated as objects and the simulation can be run. The instantiation follows the SV-1 and SV-6 very closely: each system has a line, each resource exchange has a line in the code. The reader can find the full code in Code Block B.2 in Appendix B.

While it is clear that the SV-1 and the SV-6 can provide a significant amount of structure to an agent-based model, there are gaps of knowledge that they cannot fill. For example, neither view provides the information about data generation, how

or when the systems act on data, and the goals and desires of each agent. Some of these can be inferred by the modeler but not all. For example, it is impossible to infer how the *Pilot Workstation Computer* would react after receiving *UAV Position* information. Does the *Pilot Workstation Computer* change the *Waypoint* when *UAV Position* is close to the previous *Waypoint*, or does it confirm? What other *Waypoints* does the *Pilot Workstation Computer* output and when or under what condition? Without such details a meaningful agent-based model cannot be constructed.

There are other behavioral knowledge gaps. One such gap is how long each communication takes compared to the actions. When a *Waypoint* is sent to the UAV, how long does the transmission take (from sent to received) compared to the amount of time it takes for the UAV to reach that position? Such values are significant factors when the designs of the systems are considered: if the UAV speed is much slower than the communications speed, system designers will focus on the UAV speed. For such reasons, the supplied information within these two views is not enough. Some of such values may be found in an SV-7; however, it is important to note that many of the missing parameters may not be related to the systems but the environment they are operating.

The lack of knowledge creates mechanistic problems as well. For example, every time the *Sensor Payload Workstation* receives a video file, should it generate a *Pan/Tilt/Zoom* command? If so, the communication network will be saturated with these commands because the *Sensor Payload* is creating a video file continuously. When the code runs, the *Command Datalink Ground Receiver* receives these messages continuously and has to send them out continuously. Making the situation worse is the apparent infinite loop the simulation goes into. Figure 117 depicts this loop in graphical form.

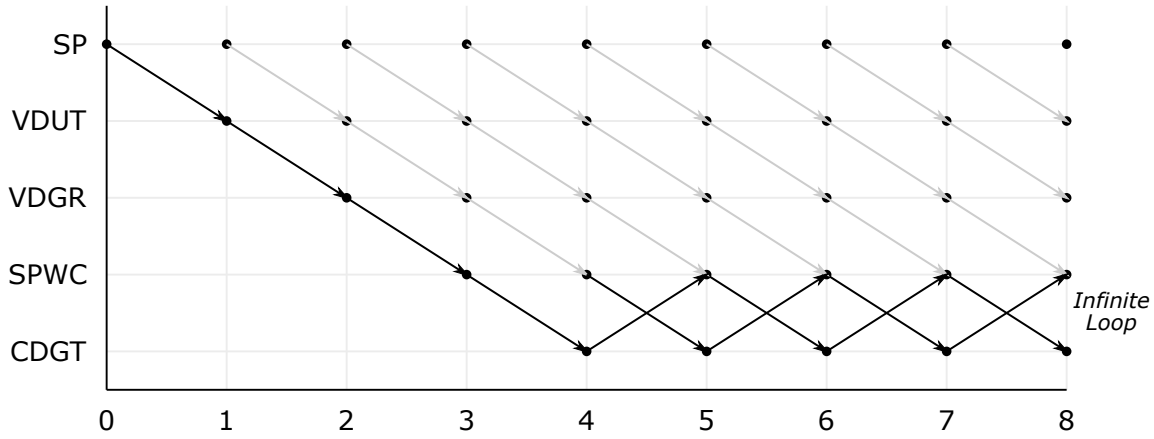


Figure 117: One of the several infinite loops that arise in the agent-based model.

After receiving the pan-tilt-zoom message, the *Command Datalink Ground Receiver* can produce a *Sensor Orientation* message back. This message triggers another *Pan/Tilt/Zoom* from the *Sensor Payload Workstation* and the cycle repeats ad infinitum. Additionally, the cycle does not remain local; it spreads to other cyclic subgraphs. Again, after receiving the *Pan/Tilt/Zoom* message, the *Command Datalink Ground Receiver* sends a *UAV Position* message to the *Pilot Workstation Computer*, which returns a *Waypoint* message and another infinite cycle starts. The reader can study the SV-1 and notice other cycles that will be trigger by this process.

The growing infinite cycles will ultimately drive the simulation to a code execution crash. One solution to this is to use sets instead of lists for the messages that need to be sent from any system. This ensures that each system submits one of each type of messages it can submit. Using sets will stop the message numbers from growing and eliminate the issue of the code crash; however, it also means that the new messages arriving at a system will overwrite the previously arrived messages. This behavior may not be valid and without more information on the system of systems in the form of other architecture views, there is no way of knowing.

Having failed to implement an agent-based model and after reviewing the reasons why the failure happened, it is concluded that the agent-based modeling is indeed

not possible with a pair of SV-1 and SV-6. These two views simply lack the necessary information a sound agent-based model must be built upon. They include no actions, no reasons for the systems to exist, why they were designed: there is not template for behaviors. These two views also do not include any information about the environment the systems are expected to operate in. In Section 8.3.2.1, a similar setup will be discussed. The conclusions are similar.

8.2.1.2 Building a discrete event model (Type B experiment)

The second experiment is to take the same two views (SV-1 and SV-6) and attempt to build a discrete event simulation. Table 190 predicts that this modeling effort is possible. The SV-1 will be used to develop the overall event network structure, the SV-6 will be used to define the evolution of the entities through the network. Because only messages are defined in these views, the discrete event simulation will be built to analyze the transmission and interpretation of the messages. The model structure will heavily resemble the agent-based implementation presented above.

Table 190 also suggests in the note section that the translation might not be possible even though the SV-6 appears to be very useful for creating discrete event simulations. The lack of confidence originates from the lack of timing information. If modeling indeed does fail, the real reason for this failure will be found and reported. This discovery can help correct the faulty parts of the hypothesis.

The discrete event simulation engine used here was developed earlier for system of systems problems where failures and timeouts are kept track of. The event timeout functionality was not important for this problem but the event failure feature was used. The timeout is useful when tracking an uncooperative target such as an enemy. When the target evades tracking for some amount of time, the timeout function drops the target from "being tracked" to "not found". While the timeout is somewhat difficult to implement in a discrete event formulation, the failure function is extremely

easy to be included and used. The simulation engine is coded in Python language and relies on libraries included in the Python's core packages to make installation on any machine easy.

The sources and sinks in the SV-6—the systems in SV-1—are modeled as servers. For this setup to work, each server had to be assigned to a number of events (minimum 1 event is required). For each server performing an event the simulation engine requires two metrics: the probability of the server performing the event successfully and the amount of time it takes for the event to be completed. The first metric is simply supplied but the second metric takes in a beta probability distribution using its 4 parameters: α, β, and lower and upper limits. These numbers are defined in the definition of events in the mechanics of the code.

The rows of the SV-6 are modeled as the events in the simulation. This means that the servers process and send messages in some finite amount of time and when the messages are sent, the events fire. The time probability distribution is not defined in either SV-1 or SV-6 and is filled in by the modeler. Such metrics would be defined in architecture views such as the SV-7. Additionally, in the DES formulation used here, the servers have the option to fail the events, e.g., a processing error occurs. The probability of this failure is also filled in by the modeler without any real data provided by the architecture. Similar to the processing time, such metrics would be given in an SV-7 but their absence do not cause the structure of the model to be undefinable. The existence of enough structure to construct models is being tested in these experiments.

Finally, a map for event execution is required by the simulation engine, i.e., what events cause what events to be executed. This is the final link in making the simulation executable. Unfortunately, in Jones Wyatt's views, there is no clear starting or ending event, i.e., arrival and departure events. Here, it is assumed that the sending of the video file is a viable candidate for a start/finish event because it is fairly isolated

and its server only performs a single event. Ultimately, in a looped network, this decision is not significant as the main study would focus on a steady state performance of the network, not the initial transition period. With this assumption, the simulation then computes the steps that is necessary for the video file to be created and transmitted to necessary systems, those systems making decisions based on the video and communicating such decisions to the UAV system so that the UAV's camera can point to different places and produce the desired video files.

This setup does execute without errors and the code works as expected. Most importantly, because there are no arrival and departure events the events loop executes forever. The execution ends after a set time to catch for infinite loops. The statistics show that the simulation end time is close to this set time and always greater. This means that the simulation ends once the next event has an execution time greater than the set time, which is expected. Because event times are random, the end times of simulations are not exactly the same but a short period after the maximum simulation time. Figure 118 shows the cumulative distribution function of the time unit when simulation ends. As the reader can see, before 100 units, there is zero probability of ending and immediately after 100 units, all 250 repetitions of the simulation end very quickly.

The simulation keeps approximately 25–30 events in the future event list during execution, as can be seen in Figure 119. Some of these future events are enqueue and dequeue events that are within the mechanics of the simulation engine being used. The list does not grow because one event must be completed before the others can be queued up. However, with the end of most events more than one event is queued up; therefore, the queues for each server (system) grow linearly in time as shown in Figure 120. This is correct from a coding perspective, i.e., the code passes verification; however, this is definitely not the intended behavior, i.e., the code is not a valid representation of reality.

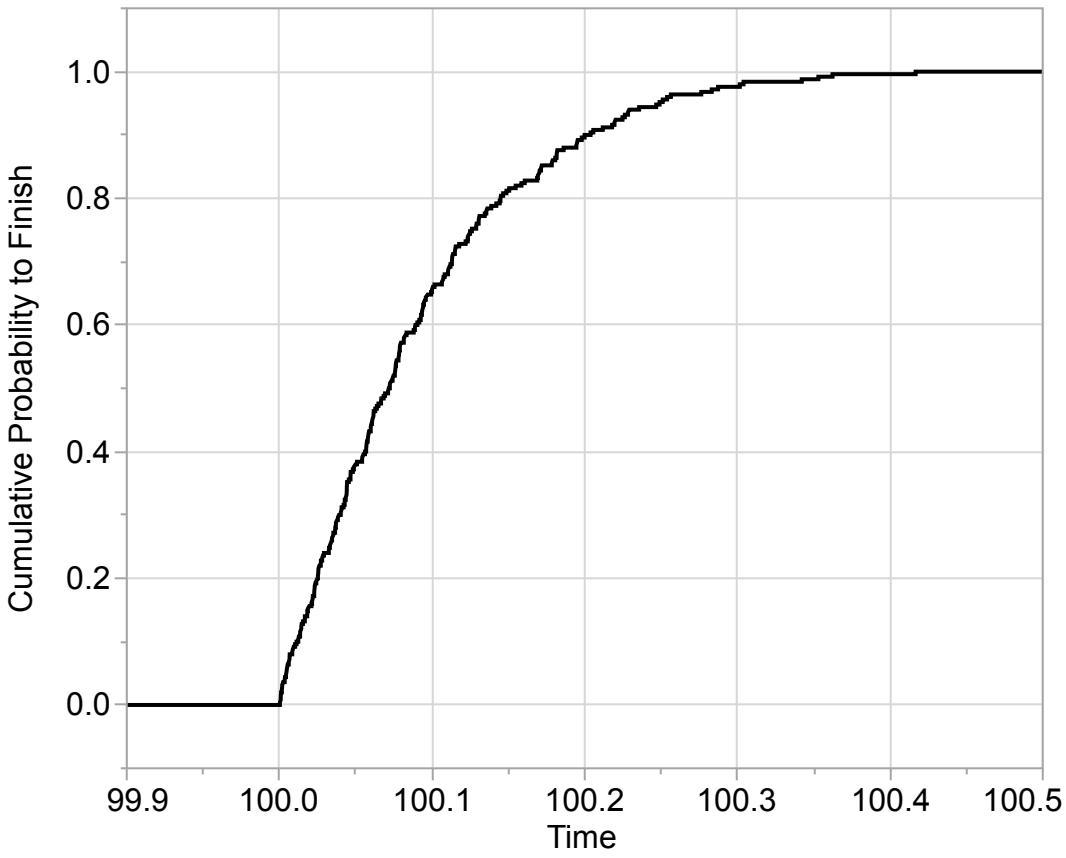


Figure 118: The simulation ends immediately after 100 time units. The data is based on 250 repetitions.

A discussion on the differences in the ABM and DES implementations is warranted. ABM experiment was made executable via a data structure workaround, whereas the DES experiment was executable on its own. Using sets instead of simple lists enabled the ABM to not get clogged up with a large number of similar incoming messages. Sets can only contain a single copy of a type of item whereas lists can contain any number of them. Therefore, if multiple messages of the same type, e.g., waypoints, the recipient will only pay attention to the last message it receives. In the DES formulation, this trick is not needed because it already includes a mechanism to manage incoming messages: queues. It is also important to note that the availability of such data structures is important in the implementation language as well as the modeler's knowledge. The implementation step is always problem and engineer

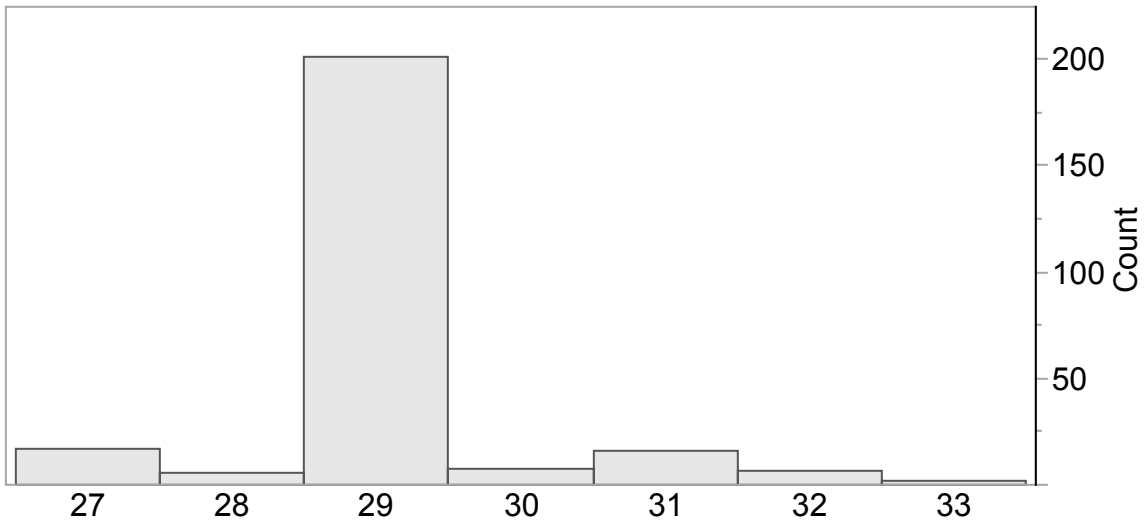


Figure 119: Roughly 25–30 future events are kept in the list thanks to the queuing behavior. The data is based on the maximum number of events in the list for 250 repetitions.

dependent.

While DES formulation works better than the ABM for this problem, neither works as intended and both require extra logic to be defined. The missing logical link is the details on what systems must do after receiving a specific signal. In the ABM formulation, this logic would be defined as internal behavior functions; whereas in the DES formulation the logic would be defined in the entity transition network and/or non-deterministic functions.

8.2.1.3 Building a system dynamics model (Type C experiment)

System dynamics models are predicted to be a straightforward implementation for the RWDC problem in Table 190. A model is attempted here after the mixed results from the ABM and DES experiments. This will be treated as a Type C experiment, and the minimum amount of architectural information is sought. Therefore, only a single view is used to start the process, and others are only added when needed.

Each row in the SV-6 was used as a flow. The flows go from source to sink as

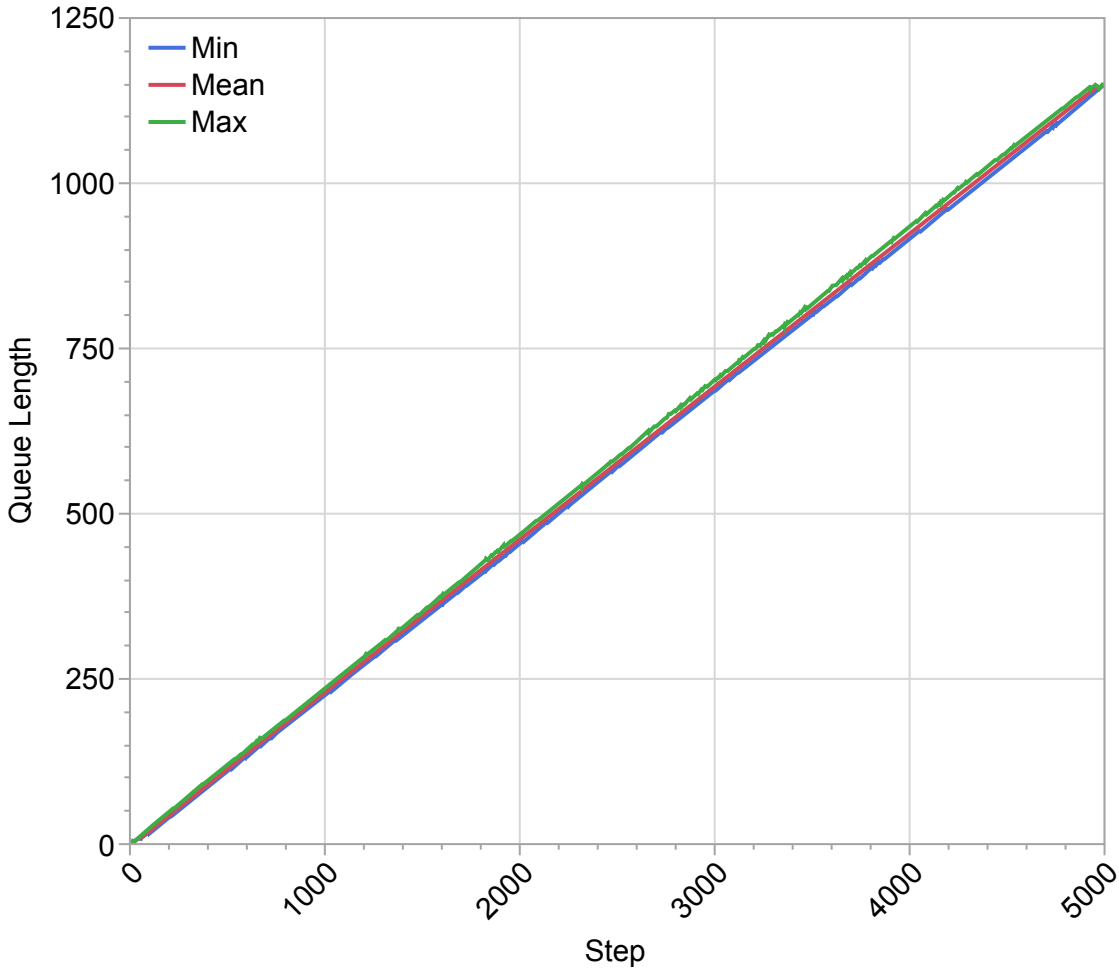


Figure 120: The queues grow linearly over time with no limit. The graph shows minimum, mean, and maximum queue lengths at each simulation step over 250 repetitions.

described by the SV-6. The sources and sinks are modeled as stocks in the system dynamics nomenclature. Each system node in the SV-1 could have been used as a stock but in the experiment, only the SV-6 was used to test the hypothesis more effectively. The resulting model can be expressed graphically, as shown in Figure 121. The entire model is then represented as a system of first order linear differential equations. Not all system dynamics models are necessarily linear, but this one turned out to be linear based on the description of the system of systems.

The final equations are given in Equations 66–73. Implementing the equations in code is fairly trivial and Code Block 8.1 shows the implementation used in this

work. The function shown in the code block is integrated using a numerical ordinary differential equation solver and plotted in Figure 122.

$$\frac{\mathrm{d}S_{SP}}{\mathrm{d}t} = f_0 - f_{15}S_{SP} + f_{13}S_{FCS} - f_{14}S_{SP} \tag{66}$$

$$\frac{\mathrm{d}S_{VDUT}}{\mathrm{d}t} = f_{15}S_{SP} - f_{16}S_{VDUT} \tag{67}$$

$$\frac{\mathrm{d}S_{VDGR}}{\mathrm{d}t} = f_{16}S_{VDUT} - f_{17}S_{VDGR} \tag{68}$$

$$\frac{\mathrm{d}S_{SPWC}}{\mathrm{d}t} = f_{17}S_{VDGR} - f_{18}S_{SPWC} + f_7S_{CDGT} - f_8S_{SPWC} \tag{69}$$

$$\frac{\mathrm{d}S_{CDGT}}{\mathrm{d}t} = f_8S_{SPWC} - f_7S_{CDGT} + f_1S_{PWC} - f_2S_{CDGT} + f_{4,6}S_{CDUT} - f_{3,5}S_{CDGT} \tag{70}$$

$$\frac{\mathrm{d}S_{PWC}}{\mathrm{d}t} = f_2S_{CDGT} - f_1S_{PWC} \tag{71}$$

$$\frac{\mathrm{d}S_{CDUT}}{\mathrm{d}t} = f_{3,5}S_{CDGT} - f_{4,6}S_{CDUT} + f_{10,12}S_{FCS} - f_{9,11}S_{CDUT} \tag{72}$$

$$\frac{\mathrm{d}S_{FCS}}{\mathrm{d}t} = f_{9,11}S_{CDUT} - f_{10,12}S_{FCS} + f_{14}S_{SP} - f_{13}S_{FCS} \tag{73}$$

```
def rwdc_model(states, time):
    """ RWDC system dynamics model"""
    sp, vdut, vdgr, spwc, cdgt, pwc, cdut, fcs = states
    #
    source_to_sp = 10
    sp_to_vdut = 0.5
    vdut_to_vdgr = 0.5
    vdgr_to_spwc = 0.5
    spwc_to_sink = 0.5
    spwc_to_cdgt = 0.1
    cdgt_to_spwc = 0.1
    cdgt_to_pwc = 0.1
    pwc_to_cdgt = 0.1
    cdgt_to_cdut = 0.1
    cdut_to_cdgt = 0.1
    cdut_to_fcs = 0.1
    fcs_to_cdut = 0.1
    fcs_to_sp = 0.1
    sp_to_fcs = 0.1
```

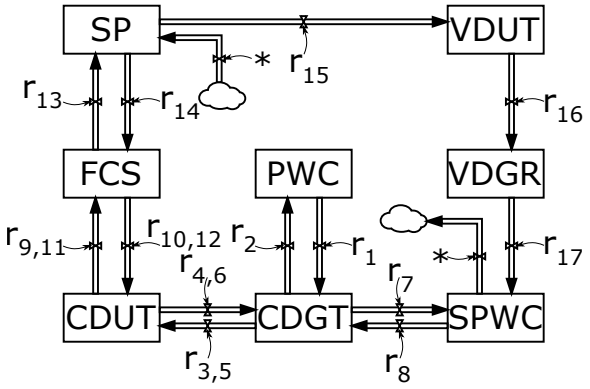


Figure 121: A graphical representation of the Real World Design Competition System Dynamics model

```
20      #
21      d_sp = (source_to_sp - sp_to_fcs * sp)
22      d_vdut = (sp_to_vdut * sp - vdut_to_vdgr * vdut)
23      d_vdgr = (vdut_to_vdgr * vdut - vdgr_to_spwc * vdgr)
24      d_spwc = (vdgr_to_spwc * vdgr - spwc_to_sink * spwc
25                + cdgt_to_spwc * cdgt - spwc_to_cdgt * spwc)
26      d_cdgt = (spwc_to_cdgt * spwc - cdgt_to_spwc * cdgt
27                + pwc_to_cdgt * pwc - cdgt_to_pwc * cdgt
28                + cdut_to_cdgt * cdut - cdgt_to_cdut * cdgt)
29      d_pwc = (cdgt_to_pwc * cdgt - pwc_to_cdgt * pwc)
30      d_cdut = (cdgt_to_cdut * cdgt - cdut_to_cdgt * cdut
31                + fcs_to_cdut * fcs - cdut_to_fcs * cdut)
32      d_fcs = (cdut_to_fcs * cdut - fcs_to_cdut * fcs
33               + sp_to_fcs * sp - fcs_to_sp * fcs)
34      #
35      return [d_sp, d_vdut, d_vdgr, d_spwc, d_cdgt, d_pwc, d_cdut, d_fcs]
```

Listing 8.1: Derivative equations for the RWDC system dynamics model

In this example it can be seen that the SV-6 provides the necessary structure to build a system dynamics model. The model is defined with stocks and flows and the missing pieces are the values of coefficients that would modify the strength of the flows. The hypothesis holds here without modification. System dynamics models out of an SV-6 are possible.

One piece that was also missing is the flow of information into each of the stocks

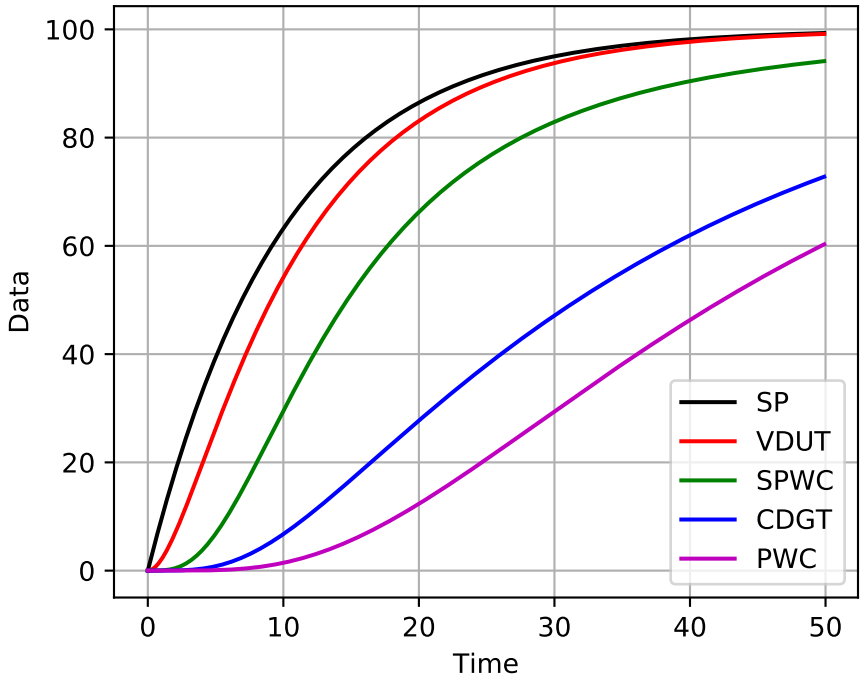


Figure 122: Selected results of the Real World Design Competition System Dynamics model. There are more stocks in the model but they are left out for clarity.

from outside the system. However, arguments can be made against failing the hypothesis for the lack of external flows in this case. It is entirely possible that the creation and consumption of "video data" was left out in the original SV-6 as architecture views are not necessarily fully complete. Jones Wyatt's SV-6 does not include the flow of data into the sensor payload. Her reasons for not including it was due to the fact her architectures are focused on communication systems. Therefore, a system dynamics modeler must fill in the gaps of creating a source for data without having to read it from the given SV-6. This simple modification is not deemed to break the hypothesis.

8.2.1.4 Review of Jones Wyatt's graph model and summary

Jones Wyatt has used architecture views to construct probability models. Her models are more detailed and specialized than the fundamental probability models described

in Chapter 5; however, the steps in their translations are similar. She takes the interoperabilities of pairs of systems and assigns a *data transfer probability* between them. This approach results in a fairly sparse adjacency matrix with probability values used instead of a simple 0 or 1 indicator value. Additionally, her communication probabilities are modified by message translations.

Finally, she splits the network into smaller networks specific to each resource which increases the number of matrices to be analyzed but reduces the complexity of each. A similar approach could be applied to the system dynamics model shown in the previous section (Section 8.2.1.3). Instead of having a large system dynamics model, multiple models can be constructed for each resource. Jones Wyatt details the split in her Table 10 in Chapter 6. Her work demonstrates how a probability model can be constructed from an SV-6 table. The reader is encouraged to review her dissertation for detailed descriptions of her models [111].

It is important to note that, while Jones Wyatt's architecture views are extremely useful in creating the models she needed for her studies, her models do not capture every aspect of the system she is describing. For example, the graph model she builds performs well for the study of interoperability and her graph models can be repurposed to calculate throughput calculations. While this single modeling type can be used for multiple purposes, different metrics must be used on the edges and this means the systems engineer must build multiple models of the same type.

Additionally, a graph model will not be suitable to study data storage issues, queueing of messages in a network, timeliness of information received, or compare physical designs of multiple designs of the UAVs (one may be faster than the other). A discrete event or agent-based model would be more suitable in such studies. And the same problem applies here: such higher fidelity models lack the abstraction capabilities to measure high level metrics such as throughput or interoperability. This impossibility is a strong support for Research Argument 2 (a sufficiently complex

system of systems will require more than one modeling technique for analysis).

8.3 2011 National Airspace System Enterprise Architecture Framework

The FAA NASEAF [19] is comprised of three separate architectures: as-is, near-term, and far-term. The multitude of models presents an interesting opportunity to compare them against each other. All three architectures were developed by the same organization[3]; however, the data available and the purpose for each version is not equal. The "as-is" architecture is mainly for documentation, standardization, and certification purposes, while the others are used for planning and development purposes. Because the purposes are different, the set of viewpoints developed also differ (significantly between as-is and others). Table 191 shows the various architectures developed for each of the three architectures.

[3]There are some significant format differences within and between the architectures and it is not possible to claim that they were indeed prepared by the same people within the FAA.

Table 191: The viewpoints developed for the three FAA NASEAF architectures.

Viewpoint	As-is	Near-term	Far-term
OV-1	✓	✓	✓
OV-2	✓	✓	✓
OV-3		✓	✓
OV-4			
OV-5a		✓	✓
OV-5b	✓[b]	✓	
OV-6a			
OV-6b			
OV-6c	✓		
SV-1		✓	✓
SV-2		✓	✓
SV-3			
SV-4		∼[b]	∼[b]
SV-5a		✓	✓
SV-5b		✓	
SV-6		✓	✓
SV-7			
SV-8			
SV-9			
SV-10a			
SV-10b			
SV-10c			
BDD[c]		✓	
TV-1[d]	✓		

[a] IDEF0 type format
[b] Hierarchical type; therefore, not useful for modeling
[c] SysML Block Definition Diagram
[d] This viewpoint is the StdV-1 but is named TV-1 in the FAA architecture

8.3.1 NASEAF As-Is Architecture

The as-is architecture consists of only operational viewpoints and a very large standards viewpoint. The standards viewpoint itself holds significantly more information than all the other viewpoints combined. It is clear that this architecture was developed for standardization and certification purposes and represents the status quo for the FAA. The StdV-1 provides significant detail on how systems within the NAS work and system views are not needed for this architecture.

Observations from the selection of the viewpoints available are as follows. The viewpoints that are not operational or systems are ignored in the analysis due to the scope of the work. Table 192 offers details on the observations listed below.

1. Even with this limited set of architecture viewpoints, all modeling types appear to be possible. With the exception of discrete event and agent-based models, all modeling types appear to be modelable using only a single viewpoint.

2. OV-5b appears to be the most valuable source of information. It has the potential to be used in any of the modeling types and in four of them, it is the most useful viewpoint.

3. The higher numbered viewpoints are critical for discrete event and agent-based models as they cannot use much of the information included in the earlier viewpoints.

Observation 2 is not surprising. At this high level of abstraction the requirements of each operational activity is extremely useful for modeling. Especially, when used with the sequence information to be included in the OV-6c it has the potential to support even the most detailed modeling types. Alternatively, when used with the higher-level organization views OV-1 and OV-2 it can determine whether the required connections are there for the system of systems to work or how likely the functions are

Table 192: Observations from the as-is version of NASEAF

Model	Feasible	Views ordered by utility	Relevant Obs.
Graph	✓	OV-2, OV-5b, OV-1	1
Probability	✓	OV-5b, OV-1 = OV-2, OV-6c	1 and 2
System dyn.	✓	OV-5b, OV-1	1–3
Markov chain	✓	OV-1 = OV-5b, OV-2, OV-6c	1 and 2
Petri net	✓	OV-1 = OV-5b = OV-6c	1–3
Queueing	✓	OV-6c, OV-5b = OV-1	1 and 3
Disc. event	∼	OV-5b = OV-6c, OV-1	1–3
Agent-based	∼	OV-6c, OV-1 = OV-5b	1 and 3

to work (or alternatively how many times they need to be executed before successfully finishing them). Similar discussions were documented in Chapter 6.

Observation 3 was also predicted in Section 6.2. The OV-5b is a much more useful source of information with increased granularity and detail. The OV-2 is really only useful for graph models and even when used, its outputs do not generate significant insight. However, it is a frequently-developed view for non-modeling purposes.

Observation 1 is an interesting observation that deserves a discussion. Taken at face value, it seems to be against the hypothesis stated in Section 3.4, which will be tested separately. At this stage the only conclusion that can be drawn is that most models are possible from a restricted set of viewpoints; there is no guarantee claimed that the models will work or even produce any useful information. The reader is reminded that just because every OV-2 can be translated into a graph, does not mean that the resulting graph is useful. It is also important to remember that just because a combination of views is possible to support the creation of a model type, it also is not guaranteed that every such combination will be enough to create that model. At this stage, the results focus on possibilities, not certainties (and given DoDAF's flexible nature, certainties may be too much to ask).

8.3.1.1 Building a discrete event model (Type B experiment)

A good place to start the experimentation on this architecture is the OV-5b and discrete event modeling because it has the highest uncertainty whether it is possible or not. Agent-based modeling for example has a higher chance of not being possible. Because there is no certainty about modeling possibility, this is a Type B experiment, as laid out in Section 8.1. Figure 123 shows a snippet from the as-is architecture's OV-5b. This view was chosen as an example because it seems to be the most useful view in the architecture. The view is prepared in the IDEF0 format and the arrows have different meanings depending on their orientation:

Down from above Rules, control, and logic

Into left Input to the process

Up into bottom Performer, mechanism through which the process is done

Down from bottom Calling another function as part of the process

From right Output object of the process

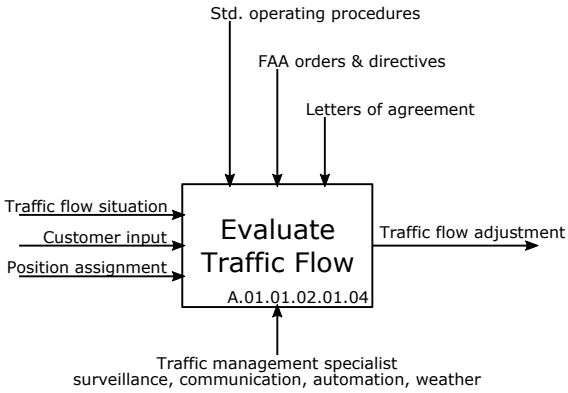


Figure 123: A part of the NASEAF as-is OV-5b

The translation of the view into an executable model is not an automatic process because the modeled can include technical knowledge and differentiate similar

architecture elements into dissimilar modeling elements by their name. For example, in the translation shown in Figure 124, there are three input arrows. However, an expert can tell that a customer input is not necessary to start the process while the traffic flow situation is always needed for the process to successfully execute.

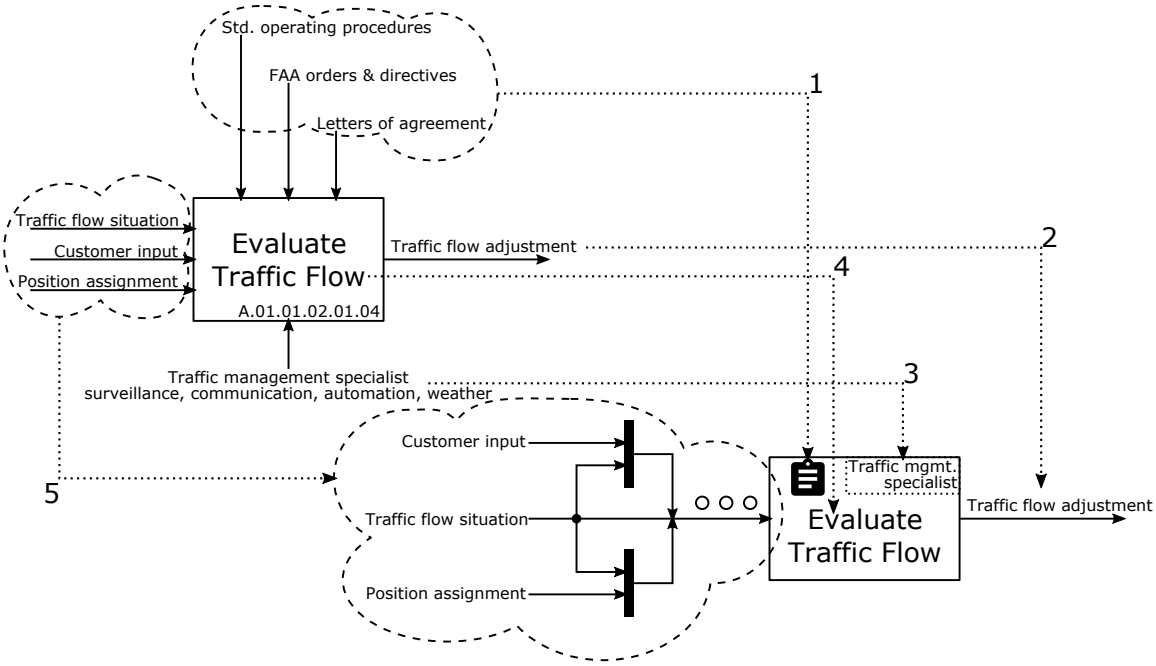


Figure 124: A possible translation of the OV-5b to a discrete event model

The example translation is detailed as follows:

1. The rules, control, and logic become part of the event. It may be used to judge how much time it takes to execute the event or what kind of output is created.

2. The output becomes the output entity of the event. This entity may carry information that makes it unique or not to make it a simple entity that can only be counted.

3. The performer is the server of the event. The mechanism could be moved into the logic for the event but could also be a performer itself, e.g., a machine that processes the entities and executes the event.

4. The function becomes the event. This is self-explanatory.

5. The inputs become the incoming queues. The entities are flowing into these queues in the larger model. The example shows a fairly complicated queue model but simple queues are also possible.

Based on Table 192 a Markov chain model could also be constructed from this view. The translation is completely different in that case as Markov chains are more simplistic models and do not have as many modeling elements as discrete event models. Figure 125 shows the translation process for this example.

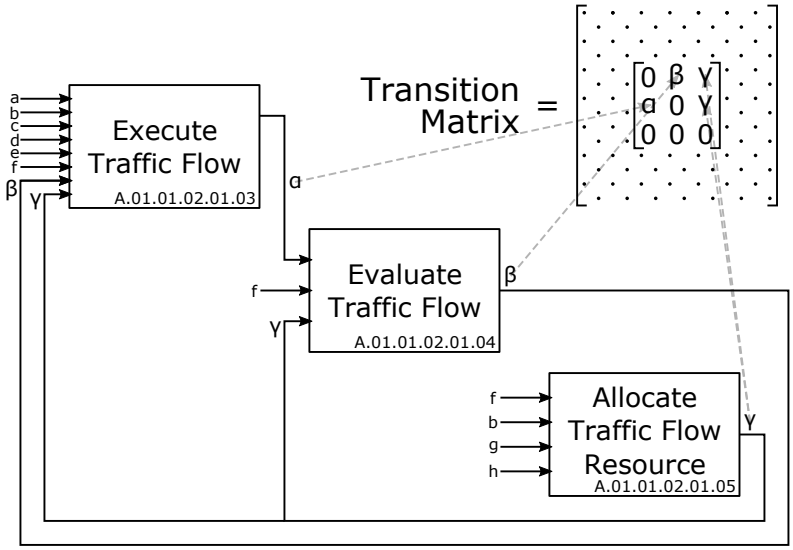


Figure 125: A possible translation of the OV-5b to a Markov chain model

Figure 125 depicts how the system will evolve to maintain traffic flow. This interpretation of the architecture view is significantly different from the discrete event interpretation. Here, the entire system transitions from one state to another meaning that there are times when the system is in the evaluation state and by definition not in any of the other states. Therefore, this model tracks the system as a whole not individual flight routes and requests. If the system can perform these functions in parallel, the Markov chain model will return erroneous results unless a single route entity is used in the discrete event simulation.

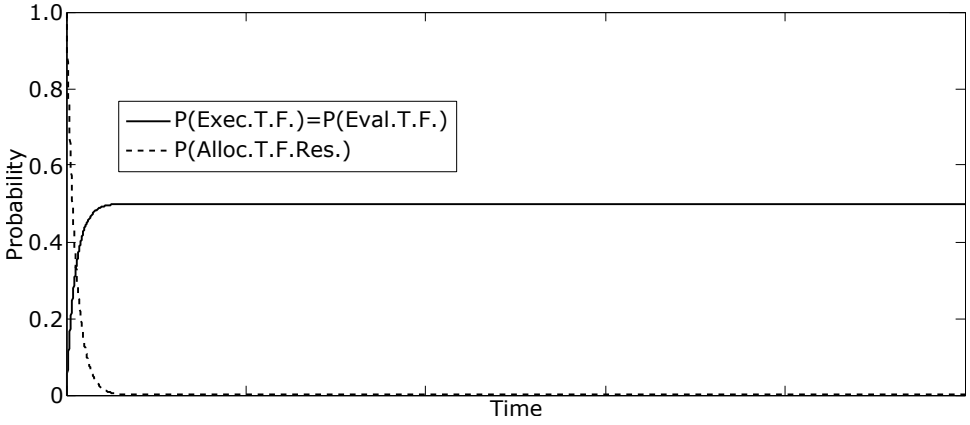


Figure 126: The resulting Markov chain output for the probabilities of each state being active.

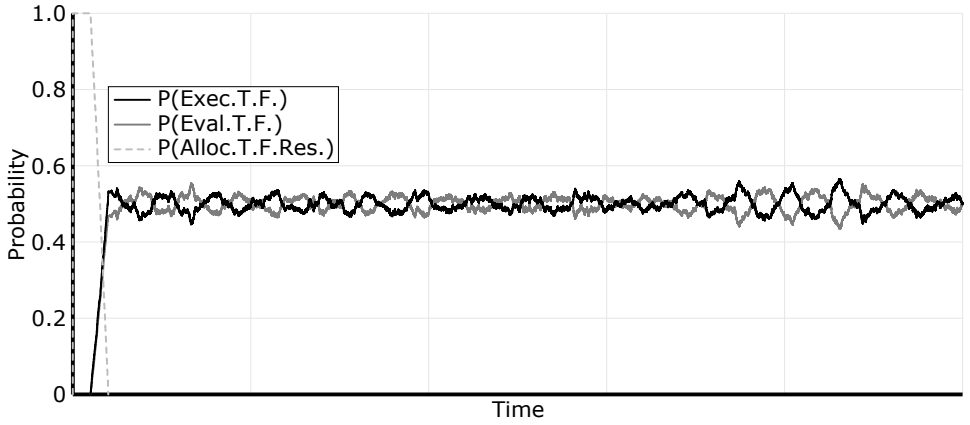


Figure 127: The resulting discrete event output for the probabilities of each server being busy.

Figures 126 and 127 show similar results for when there is a single entity in the discrete event simulation. This is done to initially cross-validate the simulation engines and models. The horizontal axis scales do not matter here because the run is only for validation of simulation types. The discrete event output is noisier as expected, because the Markov chain output is a highly accurate numerical solution to an analytical system of equations. Because the accuracy is high, the probabilities of the first two states are exactly on top of each other and only one can be shown in the figure. Both figures show that the state/event *Allocate Traffic Flow Resource* very quickly processes the work and passes it along to the other states/events. At the

steady state, the other states/events are active 50% of the time. The main source of the discrete event simulation can be found in Code B.1 in Appendix B, which does not include the various inputs to it for brevity.

Adding more entities to the discrete event model breaks result similarities between the models. In this example, the Execute Traffic Flow state/event is made to work slower than the Evaluate Traffic Flow state/event. Therefore, one remains always busy while the other stays sometimes busy in the discrete event model. Because the Markov chain can only be in a single state at any time, the probabilities must add up to one at any time as well. Therefore, as one state increases its steady-state probability the other must decrease. Also notice that the Allocate Traffic Flow Resources state/event takes much longer to decay in the discrete event model but the decay is more abrupt. Figures 128 and 129 show the differences and similarities in trends between the different simulation models.

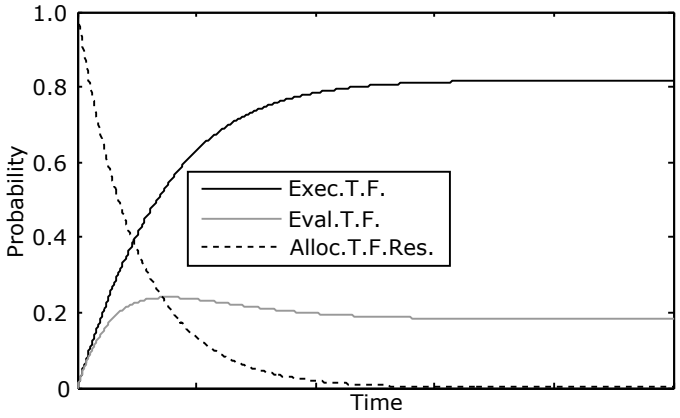


Figure 128: The resulting Markov chain output for the probabilities of each state being active.

Table 192 showed that a discrete event model is not necessarily possible from the set of views provided by this architecture. The example above is an attempt to falsify that claim and was constructed with a significant list of assumptions:

1. Operational nodes are given but information on the systems providing the functions are missing completely. Therefore, their performances or how they actually

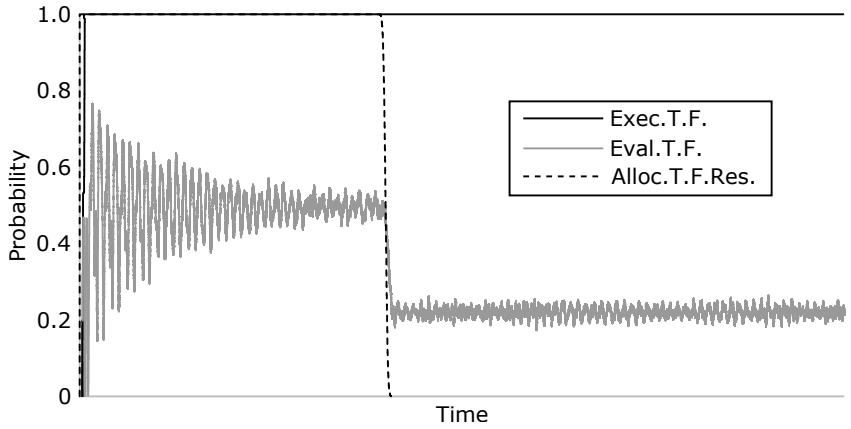


Figure 129: The resulting discrete event output for the probabilities of each server being busy.

perform the actions could not be modeled. Only a rough approximation was used: a single system that performs each event.

2. Operational nodes are nowhere near enough to describe this complex system without functions that systems provide. A list of systems was provided in the OV-1 even though OV-1 is not the correct place to publish this information. However, most of this list is paragraph descriptions of systems and not presented in a systematic way. The information is simply difficult to use.

3. Performance values are missing completely. This is not a show-stopper because the numbers are not necessary to build a conceptual model; however, the results of the simulation will not match reality unless realistic values are used.

4. The OV-5b that was used does not include information about the entities apart from their names. There is no indication whether the entities are merged inside events, whether a number of them are required to execute the event, or what happens at the splitting lines—it is not known whether entities split into two or "choose" one path or the other.

5. It is also not clear whether some incoming lines are entities at all. They may

represent some information, a conditional variable, or a simple switch to the event, meaning that they modify the event but not actually cause it.

Based on the list above, it must be said that the discrete event model example did not have enough structure to be determined as a successful model. Following the procedure for the Type B Experiment, the remaining supplied views are now investigated whether they can provide the missing information. The first view to be investigated and added is the OV-6c as suggested by Table 192. Indeed, this architecture's OV-6c includes some of the missing information, specifically, the systems[4] providing the functions (1) and how inputs and outputs[5] need to be treated (5).

The OV-6c includes a few operational examples that are described in much greater detail than the OV-5's block diagram. On the OV-6c a modeler can see every message being exchanged between systems as well as their order, meaning that a single box or a line in the OV-5 is decomposed further into sub-actions to make up that specific function. The fourth of the five examples describes the collaboration of many traffic management units (TMU) for air traffic flow planning, execution, and evaluation. These actions match to the OV-5 used above.

Figure 130 shows the OV-6c from the as-is NAS enterprise architecture. It can be seen that the consideration for weather is the first activity in the process and Air Traffic Control System Command Center (ATCSCC) is the only system that deals with it. Next ATCSCC receives user input/feedback. The third step is planning the traffic flow for the day. These can also be seen in the OV-5 of the same architecture; however, there is no order to them, nor is there any distinction between these and the possible later adjustments due to unforeseen events. The OV-5 simply shows where information comes from and where it goes; whereas, the OV-6c provides a step-by-step execution of the operations.

[4]Systems in this case would translate to servers in the discrete event model

[5]Inputs and outputs correspond to entities and queues in the discrete event model

The extraction of information from the OV-6c is not a straightforward process. The OV-5b was enhanced by highlighting the order of events using one of the scenarios given in the OV-6c. It is given in Figure 131. It can be seen that even a partial view with a single scenario is appreciably complicated. A discrete event model is then constructed to cover the allocation of traffic resources (it is a good starting point for the process at hand) and planning, executing, and evaluating traffic flow. The results are given in Figure 132.

The results show that after the initial setup the supervisors are not utilized and because major traffic flow patterns change rarely (disruptive weather events in a specific area are rare), the ATCSCC is also under-utilized. While interpreting the results the reader must remember that the numbers that were used may not be highly representative. The point of the experiment is to test the structure of the model, not the similitude of outputs to reality. Additionally, the ATCSCC is tasked with many other functions that are not covered here. The plot also shows how each center yields to the other for guidance/execution (i.e., when one is busy, the other is free). The horizontal axis scale is not important as the simulation does not end but goes into a steady state.

Returning to the question to be answered by this experiment: does this mean that the OV-5b and OV-6c are a definitively capable combination to create discrete event models? The answer must still be a "no" because even when the OV-5b and the OV-6c are this detailed, the information about the systems performing the functions is still missing. The OV-6c shows operational nodes but inside each operational node there are numerous systems in this scenario. A TMU for example is part of many different centers in the NAS. There are TMUs in ARTCCs, towers, TRACONs etc. Also, the TMU is not a single entity. It can employ a number of coordinators and therefore its capability of handling many tasks at once is a variable. The OV-6c does not include any of such information. The combination of the OV-5b and the OV-6c is

still not a guaranteed solution for a discrete event model. However, in simple scenarios where the complexity is much smaller compared to the NAS the combination can be very powerful to answer all questions a discrete event modeler may have.

The next view that may be useful in this case is the OV-1. Upon close investigation it was decided that the view does not include the missing information needed for discrete event modeling. The only figure that was provided showed a conceptual flight of an aircraft from the origin to the destination airport. The lines that connect the various nodes on it are not defined well enough to convey how the collaboration works, or how resources are shared which can be modeled by a discrete event model. Therefore, the result of the experiment is not to falsify (i.e., to confirm) the prediction that the combination offered is not necessarily enough for discrete event modeling. The current experiment is concluded here.

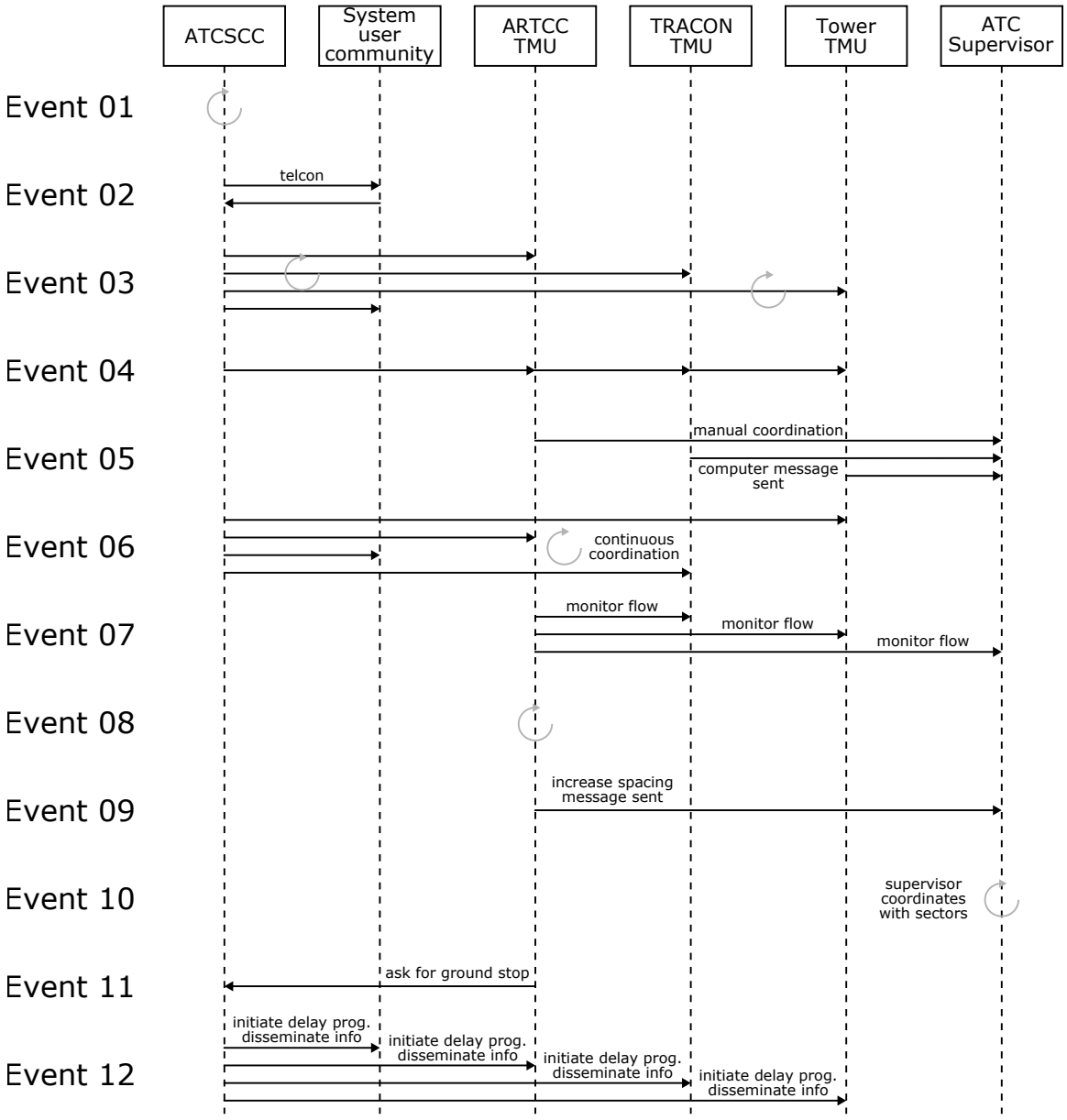


Figure 130: FAA's As-Is OV-6c (fourth out of five views)

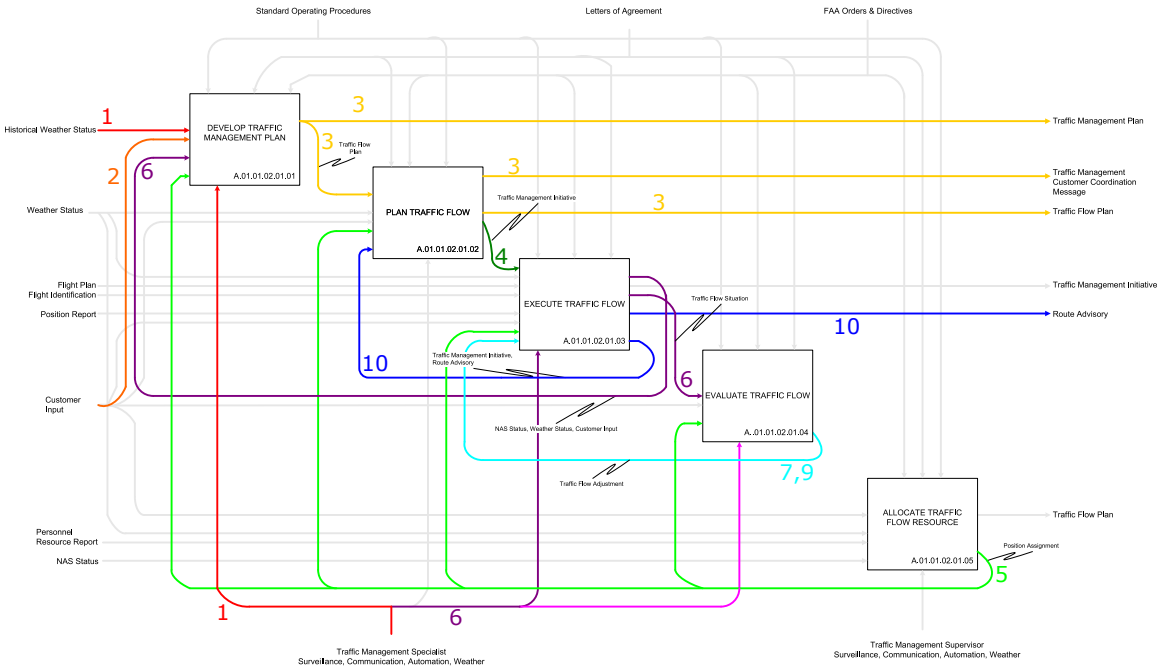


Figure 131: FAA's As-Is OV-5b with added sequence information from the OV-6c (adapted from the FAA architecture [19])

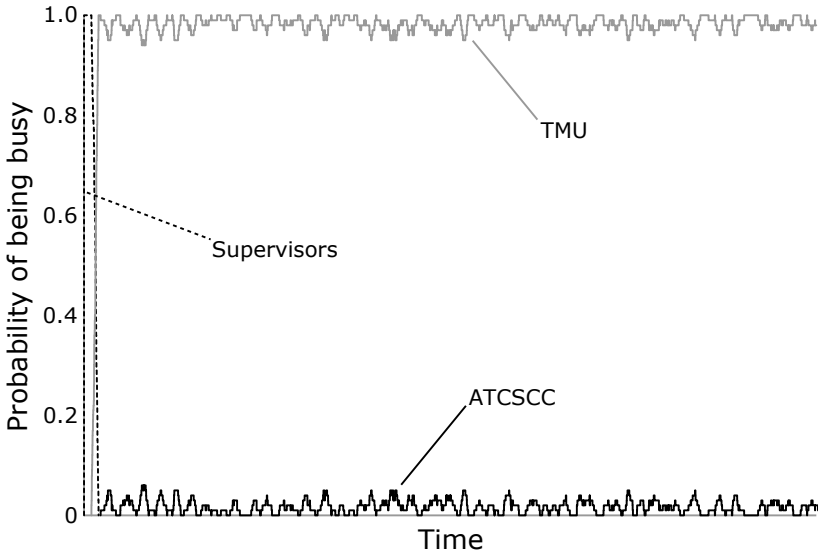


Figure 132: FAA's As-Is OV-5b with added sequence information from the OV-6c (adapted from the FAA architecture [19])

8.3.1.2 Building an agent-based model (Type B experiment)

Another Type B experiment suggested by Table 192 is building an agent-based model, which may be possible but not necessarily so. The same approach will be taken as the previous Type B experiment with a special exception. Among the modeling types investigated here there are none that are similar to agent-based models. While it was possible to compare a Markov chain to a discrete event in the previous experiment, a similar comparison is not possible here. Therefore, the cross-calibration step of the Type B experiment will be skipped. Table 192 shows that the OV-6c is the most important view to be used for an agent-based model in this case. For sake of continuity, the same subset of OV-6c used in the previous experiment will be used.

The model was created using Mesa, a Python package for agent-based modeling [132, 113]. It offers a convenient compartmentalization of modeling, analysis, and visualization. In this particular case, only the modeling part was used because the aim of the experiment is to show a model can be built not that its outputs match reality. The numerical inputs to the model had to be fabricated because the specific OV-6c given inside the FAA As-Is architecture did not include them. While the statistical analysis step was not needed, the simulation was nevertheless visualized in an OV-6c-like graphic that shows how the simulation was executed from start to finish. The reader is encouraged to compare these results given in Figure 133 with the OV-6c given in Figure 130. The model's source can be investigated in Code B.3 in Appendix B.

It appears that the model building was a success apart from the lengths of activities or whether they can run in parallel or not. These types of information could have been easily depicted on the OV-6c using the bars such as the ones shown in Figure 133. Because the original OV-6c lacked them, the model was created with made-up numbers just to prove the concept and the model executed successfully.

The main barrier however was not the lack of numbers but the lack of messages

or triggers to activities. What causes Event 4 vs. 6 was not clear on the OV-6c. The messages that were depicted were almost universally top-down with little feedback. Events that required coordination or agreement before moving forward must generate some feedback and possible back-and-forth communication. Instead of using these triggers, the model unimaginatively executes events based on a schedule and therein lies the reason why agent-based modeling from an OV-6c is not a guaranteed endeavor.

The OV-6c view depicts a specific scenario. In the view used here, the scenario was a normal start to the day for air traffic control that soon gets disrupted with inclement weather. While it is theoretically possible for all possible interactions between operational nodes as shown on a sequence of OV-6c diagrams, it is not practical and would be very difficult to synthesize all of them into a single model. OV-6c's descriptions are not as general as the other operational views.

Ideally, an agent should act on the stimuli from the environment or from agents and perform a set of reactions based on its understanding of what is going on. The simplest example to this is an if-this-then-that automaton. However, the model put together could not implement many of the interactions this way. Certain actions were simply started by the end of another or scheduled ahead of time. There is no emergence of order from the agent interactions, behavioral patterns due to agent rules, etc. The model simply schedules events to match the OV-6c and that is not useful as no new information can be gleaned from the model outputs.

The discussion cannot rule out the possibility of creating an agent-based model of a simple system with a few possible scenarios that can be exhaustively depicted with a number of OV-6cs. Therefore, the "possible but not guaranteed" label remains unchanged. Next, the knowledge from the OV-5b will be added to the mix and it will be ascertained whether it will fill the information gap or not. Table 192 suggests the OV-1 as well as the OV-5b could be useful in this scenario. However, the OV-5b is preferred over the OV-1, because it shows the inputs to functions, which are modeled

directly in the agent-based model. These inputs can be used as triggers by assuming when all inputs to a function is present, the function is triggered. This formulation may just work to fix the pre-determined schedule problem.

The OV-5b adds a significant structure to the model. Even actions that have been left out in the OV-6c can be found in the OV-5b represented by several functions and input output relations that correspond to a single process on the OV-6c. Additionally, the OV-5b includes strict rules of what functions are needed to produce a specific information and what each function does after the receipt of a specific information type. The agent-based model was then altered to not schedule actions ahead of time but have agents take in information, based on which perform an action, and finally output an information. The results are given in Figure 134 and are slightly different. However, the main point here is that the model executes agents in a more generic way acting to stimuli themselves rather than the simulation engine prescribing actions to them. Therefore, the resulting agent-based model is more structured and closer to a complete solution that the model created with just the OV-6c was.

It can be seen that the agents take some time to act on the information that they receive. This is due to the implementation of pure randomness in what agent acts first during a given time step. Additionally, some events have moved up or down based on the functions they require given by the OV-5b. Additional functions that were not explicitly shown on the OV-6c have been added to various agents because their output is required by consequent functions. For example, Event 5 now requires an ATC Supervisor to perform a final step because based on the OV-5b it is not a simple message that gets sent to the supervisors, but the supervisors act on it. This problem could have been avoided if the OV-6cs were more detailed; however, the inputs and outputs of each activity would still be represented in the OV-5b. Therefore, it can be said conclusively that either view is enough on its own to construct an agent-based model. This conclusion supports the arguments in Sections 6.6.8 and 6.9.8.

There are still two pieces of the agent-based modeling puzzle missing from the OV-5b and OV-6c pair: the environment and the specifics of the actions. Simple environments may be represented as other agents to solve the first problem but this approach is not guaranteed to work with environments that are highly variable based on location. Representing seas, mountains, or clouds as systems can also be more confusing than helpful. The specifics of actions on the other hand can be included in an OV-6a easily. The specifics outline what the action really does apart from the information it generates or how long it takes (e.g., how does an airplane fly a route, how does a missile avoid detection). These missing information are fundamentally different from the missing numerical information such as how many of each agent, or how much resource is being used. The numerical information is required at run time but is not required to construct a model.

For a fully featured agent-based simulation, this information should be included. Based on these arguments, it must be conceded that the OV-5b and OV-6c pair is still not enough for an agent-based model's construction. The result agrees with the hypothesis.

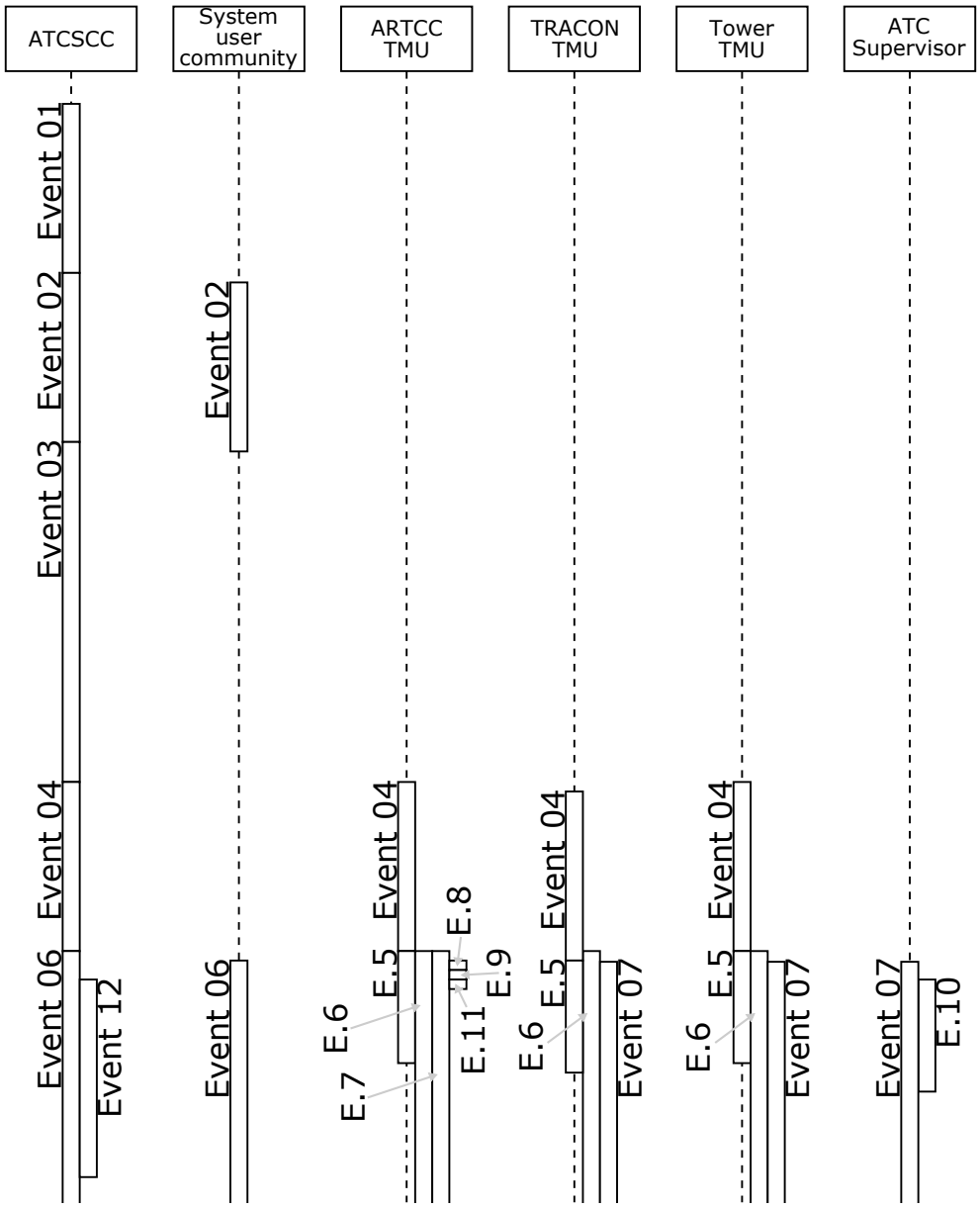


Figure 133: Agent-based model results using the FAA As-Is OV-6c

Figure 134: Agent-based model results using the FAA As-Is OV-6c and OV-5b.

8.3.1.3 Building a graph model (Type C experiment)

The possibility of creating a graph model is tested. The claim is that a graph model is easily obtainable from an OV-2, OV-5b, and OV-1; therefore, the experiment will fail if graph models are not obtainable from the trio of views. The first view to be used is the OV-2 based on the preference the table provides.

Unsurprisingly, the view is very suited for graph modeling, as discussed in Section 6.2.1. In this experiment a different diagram is used from the Section 6.2.1 example to minimize overlap. The third diagram that was provided in the FAA as-is OV-2 is given in a simplified form in Figure 135 and the adjacency matrix that encapsulates the graph model is given in Table 193. Here the number of message types exchanged between the nodes are interpreted as the weight of the edges but other interpretations are also possible. With the selected interpretation, the resulting graph is not symmetric.

Table 193: An OV-2 viewpoint from the FAA as-is architecture translated to an adjacency matrix

	RMES	ATCT	TRACON	ARTCC	OCC	NOCC	FSS/AFSS	FICO	NNCC
RMES		5	5	5	8		5		
ATCT	6		5	4	6		5		
TRACON	5	6		6	5		5		
ARTCC	5	5	6		3		6		
OCC	5	2	4	4		6	4	2	
NOCC					4			1	2
FSS/AFSS	5	6	5	6	4				
FICO					2	2			
NNCC						2			

Zeroes are left empty for easier readability.

Alternatively, one might make one graph for each information type exchanged between the nodes. Table 194 below shows the resulting adjacency matrix for the "NAS maintenance schedule" information. Notice that the edge weights here are

exclusively 0 or 1, meaning that they simply symbolize the existence or non-existence of the edges. As before, the construction of this matrix is fairly straightforward from the OV-2.

One problem a systems engineer will note when looking at this adjacency matrix is that this particular message never gets sent to the node TRACON but is sent to others from it. That will raise a flag for viability of the system to work. Turns out that other pieces of information such as infrastructure maintenance schedule or adjustments to the schedule are sent to TRACON and they are transformed into the maintenance schedules inside the node. This is a simple example of how a graph model may be used in the system of systems context.

Table 194: An OV-2 viewpoint from the FAA as-is architecture translated to an adjacency matrix

	RMES	ATCT	TRACON	ARTCC	OCC	NOCC	FSS/AFSS	FICO	NNCC
RMES					1				
ATCT	1				1				
TRACON	1	1							
ARTCC									
OCC	1			1			1		
NOCC					1			1	1
FSS/AFSS	1	1							
FICO									
NNCC						1			

Zeroes are left empty for easier readability.

From the two examples above, it is clear that just the OV-2s are sufficient to create graph models. The reader can look at the OV-1 and OV-5b, then use the similarities to imagine how they can be used to create graphs, but for the purposes of this experiment, the results above obviates the need to construct them. The claim is shown to be unconditionally correct.

8.3.1.4 Building a Markov chain model (Type C experiment)

As a final experiment for the FAA as-is architecture a Markov chain model is attempted. Table 192 suggests OV-1 and OV-5b as starting points. The OV-1 depicts a typical flight of an airliner from one airport to another and being in contact with various systems along the route. Unfortunately, while the states are shown, the transitions between them are hardly defined. Additionally, the connections do not include what kind of information is being transferred. Therefore, the focus of this experiment will be on the OV-5b. This does not mean the OV-1 is universally useless for Markov chain modeling, only that it is for this very specific case.

The OV-5b to Markov chain translation was briefly discussed in Section 8.3.1.1 as a comparison to the discrete event model. The experiment here carries on from that discussion. The missing two states and their transitions are added to that continuous time Markov chain model. The transition matrix is given in Table 195 and the resulting state transitions are shown on Figure 136.

Table 195: Transition matrix made from the OV-5b of FAA's as-is architecture

	Develop	Plan	Execute	Evaluate	Allocate
Develop	−6	0	2	0	5
Plan	6	−4	3	0	5
Execute	0	4	−7	9	5
Evaluate	0	0	2	−9	5
Allocate	0	0	0	0	−20

The values here are arbitrarily assigned, but the numerical information is not required for model structure as discussed before.

While the resulting Markov chain is not a realistic model, the experiment still validates that that conversion is possible. The inclusion of the other OV-5b viewpoints and the OV-1 context undoubtedly improve the realism of the model. However, for the

purposes of this experiment, the conversion is shown to be viable and the argument within Table 192 is supported. FAA's near-term architecture is used for the next few experiments.

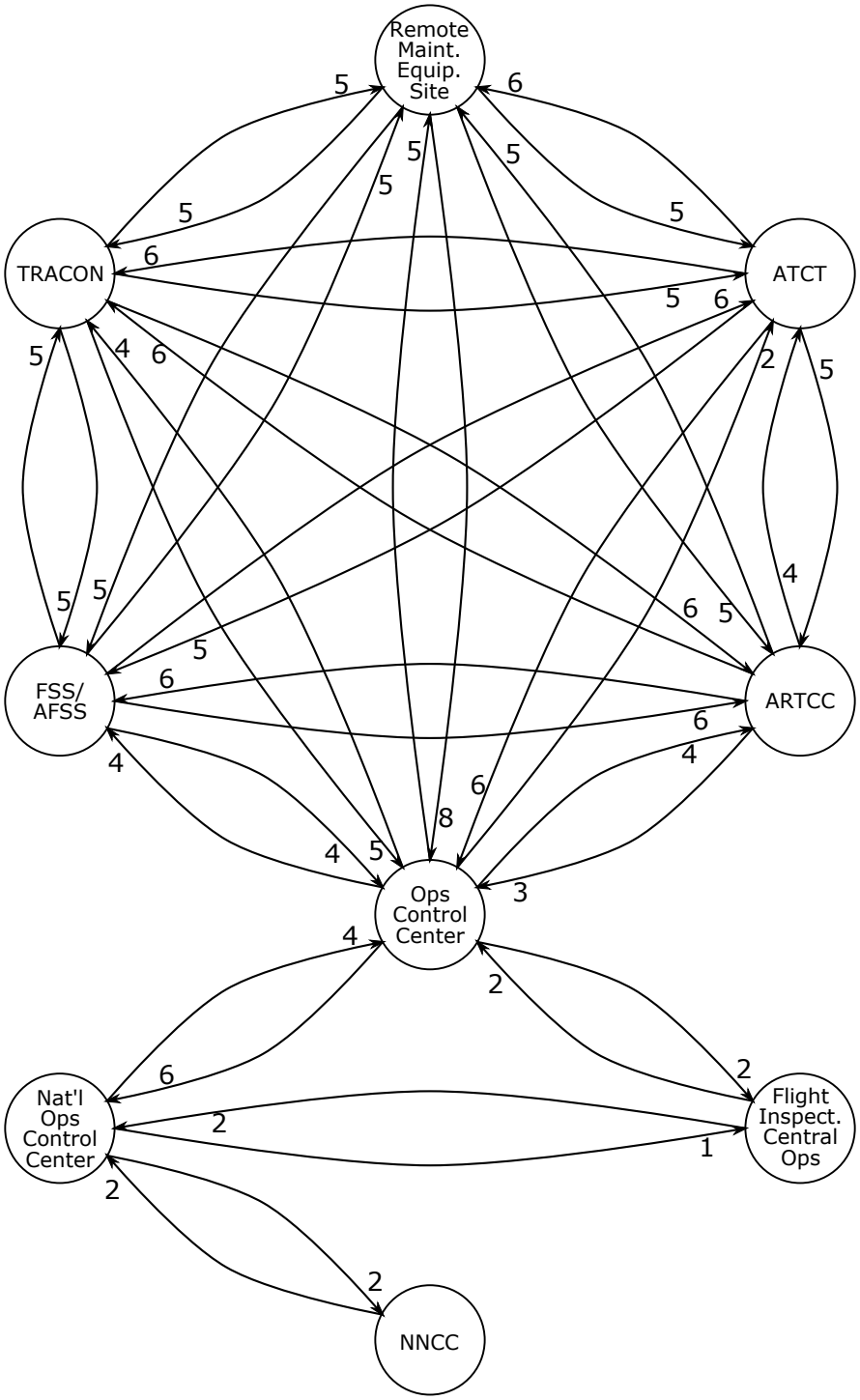


Figure 135: FAA's As-Is OV-2 (third out of three views)

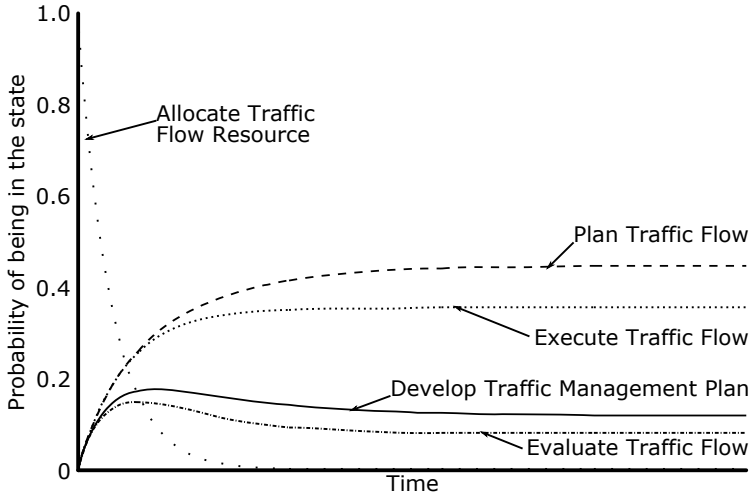


Figure 136: The resulting Markov chain output for the probabilities of each of the five states being active.

8.3.2 NASEAF Near Term Architecture

The near-term architecture adds OV-3, OV-5a, SV-1, SV-2, SV-4, SV-5a, SV-5b, and SV-6 on top of what was available in the as-is architecture, but omits the OV-6c and StdV-1. It also adds a fit-for-purpose view that is a large collection of SysML Block Definition Diagrams that thoroughly define the operational activities in great detail. The new views set a vision for the evolution of the NAS in the near-term. They are less strict than the StdV-1 supplied with the as-is architecture.

The near-term architecture's SV-4 is a hierarchy view. This type of the SV-4 is not particularly useful for modeling purposes, as discussed in Section 7.4. Because of this, the SV-4 is ignored in the analysis and observations that follow.

The jump from the as-is to near-term architecture is quite substantial. The additional viewpoints make it possible to increase modeling complexity as well as quantity, i.e., many more low-complexity models can be created from the supplied views, each of which may provide different insights in how the system of systems will function. The observations are listed below. The data the observations were derived from is given in Table 196.

1. Agent-based models cannot be created from a single viewpoint. All other models appear to be feasible from a single viewpoint (different viewpoint for each modeling type).

2. The OV-5b, SV-2, and SV-6 appear to be the most useful for almost all modeling types. These viewpoints show a process, physical network enabling the process, and what exchanges the network is used for. Many modeling types can be supported by this group of viewpoints.

3. None of the higher detail viewpoints (OV-6a–c and SV-10a–c) are used.

4. The SV-5a,b are essentially ignored because they are not useful individually.

Table 196: Observations from the near-term version of NASEAF

Model	Feasible	Views ordered by utility	Relevant Obs.
Graph	✓	OV-2 = OV-3 = OV-5b = SV-1 = SV-6, OV-1 = SV-2	3 and 4
Probability	✓	OV-5b = SV-2 = SV-6, OV-1, OV-2 = OV-3	2–4
System dyn.	✓	OV-5b = SV-2, SV-6, OV-1 = SV-1	2–4
Markov chain	✓	OV-1 = OV-5b = SV-2 = SV-6	2–4
Petri net	✓	SV-2, OV-1 = SV-4, SV-3 = SV-6	3 and 4
Queueing	✓	OV-2, SV-6, OV-1	2–4
Disc. event	✓	SV-6, OV-5b, SV-2	2–4
Agent-based	∼	SV-1 = SV-2, SV-6	all

Observation 1 is expected as no view outside of the high-detail views can support a modeling type as complex as agent-based modeling. Nevertheless, with the help of the SV-2 and SV-6 together, it may be feasible to put together a simplistic agent-based model for this architecture. Subsection 8.3.2.1 details the attempts at building an agent-based model.

Observation 2 is not surprising because the details on how systems work internally or collaborate/compete with each other is not specified on any of the viewpoints developed. Much of the information that can be derived from the combination of the OV-5b, SV-2, and SV-6 can be included in an SV-10c; however, that view was not developed for the near-term architecture, as stated in Observation 3. Therefore, in order to gather the same information, multiple views are used.

Observation 4 is important because, as discussed in Chapters 6 and 7, the results focus on the utility of individual views to discover links between the architecture views generated, and models that will fit the problem best. The SV-4, SV-5a, and SV-5b need to be considered as a unit. When one is lacking from the architecture, the others cannot be effectively used in the modeling efforts. In the case of the near-term architecture, the SV-4 is effectively missing. It must be noted that there is an SV-4 in the architecture; however, it is a hierarchy type SV-4. As discussed several times

before, the hierarchy type is not very useful for modeling. Effectively, this means that the SV-5a,b are neglected for modeling in the FAA near-term architecture. The combined use of SV-4, SV-5a, and SV-5b is not investigated.

8.3.2.1 Building an agent-based model (Type B experiment)

Table 196 suggests that an agent-based model may or may not be possible with the given viewpoints included in the near-term architecture. Therefore, an agent-based model construction is attempted as a Type-B experiment. The near-term architecture is significantly more detailed than the as-is architecture and only parts of the viewpoints will be used to create models. The inclusion of all the other layers not considered here will not fundamentally alter the model; it will only make it more generally applicable. Table 196 suggests the SV-1 and SV-2 as good starting points. Additionally, the SV-6 was used as a check between the other two views as well as to provide some extra information such as the nature of the exchanged messages.

The SV-1 shows the resources being exchanged between system nodes, whereas the SV-2 shows the interfaces that facilitate the exchanges. In the FAA architecture, the resources shared are exclusively information messages. This experiment will focus on the management of weather information that includes collecting, distributing, space/surface/aloft weather information from ground, space, and airborne sensors. "Maintenance of this information includes validating the information and the sources when generated by external stakeholders, maintaining the currency of the information (including purging expired information), producing products that result from filtering and combining different pieces of information, providing persistence of the information at various points of use, and distributing the information either on demand or according to business rules" according to NASEAF AV-2.

The model created uses 4 agent types and 1 class for data. Below is a description of what the agents do for each simulation loop.

- Systems do
  - Receive data
  - Generate data
  - Act on the data they generate or receive
  - Send data
- Interfaces do (depending on their type)
  - Transfer data directly between two systems
  - Transfer data between a system and a communication service
- Communication services do
  - Transfer data between two interfaces
- System nodes do not perform an action in this model. They are only included for structure purposes. Their role can be expanded by subsequent use of other views

Table 197 shows where the various information for the model definition originate from. There is significant information overlap that makes cross-validation useful. In fact, there were a few cases in which the overlap was used to correct mistakes in the architecture viewpoints such as interfaces having the wrong codes and systems not appearing in system nodes. Trying to construct models from the architecture viewpoints can in turn increase the quality and consistency of information included in the architectures as well. Of the three viewpoints, the SV-2 was the least detailed and sometimes the information on it did not match the SV-1 and SV-6. Additionally, the SV-2 did not show which communication service was used by each of the interfaces; it showed only whether each system node has a connection to various communication

services. The lack of detail causes ambiguity and the model constructed ended up being less specific than the potential offered by the viewpoints.

Table 197: The origin of the information used to build an agent-based simulation

Information	SV-1	SV-2	SV-6
System nodes	✓	✓	∼
System node codes	✓	×	✓
Systems	✓	✓	∼
System codes	✓	×	✓
Interface	✓	×	✓
Interface codes	✓	×	✓
Comm. services	×	✓	×
Data	×	×	✓

✓ The information was taken from this view
∼ Some of the information was taken from this view but other views had to fill in the gaps
× The information was not found in this view

The information gathered from the views was only partially enough to construct an agent-based model. All systems and system nodes as well as interfaces and communication services were modeled as agents; however, only systems reacted to input and provided some agent behavior. Even the systems simply passed on information they received to the other systems that they were in contact with. There was no internal process to do something with the information gathered. The reason for the lack of interesting behavior was the lack of details of what each system, or system nodes, or interfaces do. For example, as can be seen in Code 8.2, the system nodes really do not do anything during the execution of the simulation (their step function is entirely empty) other than being a container to systems.

```
1 class SystemNode(Agent):
2     """Shaded boxes on SV-1 or SV-2"""
3     def __init__(self, name, model, interface_identifier):
4         super().__init__(model.create_agent(self), model)
5         self.name = name # system node names must be unique
6         model.add_system_node(self)
7         self.interface_identifier = interface_identifier
8     def add_system(self, system):
```

```
9           setattr(self, system.name, system)
10      def step(self):
11          pass
```

Listing 8.2: Definition of a system node

As mentioned above, weather information generated at the weather facility is used as a case study. In each simulation step, the systems with some received information check whether the information is new to them. If the information is old news, they discard it. If the information is new, they add it to their knowledge, process it, and send it to other systems that they are connected with. In the next simulation step, those systems perform the same actions with the information they receive. Therefore, the systems have a more defined simulation step function as can be seen in Code 8.3. The code listing shows only the simulation step actions not the definition of the entire agent for brevity.

```
1       def receive_data(self, data):
2           logstring = [" ".join([self.name, "received data from",
3                                  data.sending_system_node.name,
4                                  data.sending_system.name]) + "."]
5           if data not in self.knowledge:
6               logstring.append("It was new info and added to the knowledge.")
7               self.knowledge.append(data)
8               self.process_data(data)
9           else:
10              logstring.append("It was old info and was discarded.")
11          self.model.log.append(" ".join(logstring))
12      def process_data(self, data):
13          self.model.log.append(" ".join([self.name, "processed the data"]) + ".")
14          #do something with the data (outside of the scope for the experiment)
15          self.send_data(data)
16      def send_data(self, data):
17          data.sending_system_node = self.system_node
18          data.sending_system = self
19          for key, interface in self.output_interfaces.items():
20              names = key.split("-")
21              receiving_system_node = self.model.system_nodes[names[0]]
22              receiving_system = getattr(receiving_system_node, names[1])
```

```
23              data.receiving_system_node = receiving_system_node
24              data.receiving_system = receiving_system
25              receiving_system.incoming_data.append(data)
26              self.model.log.append(" ".join([self.name, "sent the data to",
27                                              receiving_system_node.name,
28                                              receiving_system.name]) + ".")
29      def step(self):
30          for data in self.incoming_data:
31              self.receive_data(data)
```

Listing 8.3: Definition of a system's simulation step

The `receive_data` function (lines 1–11) simply takes the data that is sent to the system by other connected systems. The information for the connections mainly comes from the SV-2. Some missing connections and multiplicities are deconflicted using the SV-1 and sometimes the SV-6. The data carried on the connections are defined by the SV-6 alone. The checks for whether the information is known already are the choice of the modeler's and are not defined by the architecture. An SV-10a would be the view of choice to define such logic.

The `process_data` function (lines 12–15) is mostly empty. This is due to the architecture not providing details on what the systems actually do. System functions are listed in the SV-5b; however, the view was not considered to be very useful by Table 128. The reason for this is that it only provides names of the system functions and leaves out the details about what the functions really are. If the OV-5b were included in building the model, the agents would have a large number of empty functions in their code. Therefore, it was left out. Without a large number of SV-10as to define what the functions are, their names themselves are not useful.

The `send_data` function (lines 16–28) holds most of the behavior of the agents. Each new data received is routed to all the other systems that are connected to the system (the direction of the connection matters as well). These connections were gathered mainly from the SV-2 with a few additions and fixed by using the SV-1 and

the SV-6. It is entirely possible that the system sends the information back to the system that sent it in the first place. This is not a problem because that system will realize that the information is not new and will discard it accordingly.

The SV-1 Interfaces and Nodes, SV-2 Ports and Flowlines, and SV-6 Resources describe only interaction rules in this architecture. The roles that the systems play are left entirely undefined. Their internal processes are also not described in any detail. An expert in the systems making up the system of systems can easily figure out the missing logic needed to complete the model however. The FAA mid-term architecture's SV-1, 2, and 6 are also based entirely on communications. It is possible to include sensor data so that the environment can "communicate" with the agents.

In conclusion, while it is possible to remove the environment-agent interaction deficiency from this specific example, it seems very unlikely that the internal processes of the agents be included in the SV-1, 2, and 6 combo. Unless the agents are extremely basic automatons, this combination appears to be unable to cater for agent-based modeling. The entries for *rules* for SV-1, 2, and 6 are modified in the final table to reflect that fact. The modified table remains to be a valid tool to test the research argument however. The failure discussed in this section does not imply the failure of the research argument. The final tables can be found in Appendix A.

8.3.2.2 Building a Petri net model (Type C experiment)

The second experiment will be of Type C, out of necessity. Table 196 does not predict any impossible modeling types; therefore, no Type A experiment can be performed. In this experiment, a Petri net model is attempted. It is predicted by Table 89 that an SV-2 is a perfect fit for a Petri net communications model. Petri net models describe the transformation of things between their possible states such as a carbon cycle as atoms of carbon are bound to various other atoms forming different molecules in different places on Earth (hydrocarbons, to carbon dioxide, to carbon, etc.). The

main focus of this Petri net model is the transformation of information as it travels through the communications network described by the SV-2.

Studying the same weather SV-2 model as the previous agent-based model, a problem was noticed: the view depicts how communications are realized (IP Services, NAS Enterprise Messaging System, Dedicated Telecommunications Services, ADS-B network, etc.). However, the actual messages that are transfered are not detailed. The SV-2 does not have to include the information about the resource being exchanged, only the methods of exchanges. Naïvely assuming any nodes connected by the same exchange method exchange any message results in a very unrealistic model. For example, the weather facility communicates with the NEMC not with other nodes that are connected to the IP Services resource exchange. Therefore, any weather information generated in the weather facility must travel through the IP Services, reach the NEMC, and then be distributed from the NEMC through IP Services or NAS Enterprise Messaging Service to ATCT, TRACON, ARTCC, etc., and in turn be relayed to the Aircraft. This is due to the fact that raw weather information may not be useful to each of these system nodes, as they need the information to be in a specific format to act on it. An aircraft, for example, is given an alternate route to navigate around a storm, or an airport to divert to. Using just the SV-2 hides such nuances and makes it look like the Weather Facility can communicate with the Aircraft directly so the Aircraft can act accordingly. This is not realistic.

In order to add realism to the model, the SV-6 was used. The model is then created using the messages that go from one node to another possibly changing their states. The model starts with every node operating under the assumption of clear weather. As the weather turns with some probability, the weather facility realizes the problem with the use of its radar node (if a new more advanced radar is installed, the transition probability can be tweaked). For demonstration simplicity only the switches from clear to inclement weather are implemented except for the weather

states which can go back and forth. The other tokens are stuck in inclement weather places in this example. It is of course a trivial task to include further transitions that can put them back into the clear weather places; however, it makes for a complicated and confusing diagram to communicate the idea of the model.

The result is shown in Figure 137. It is one of the many alternative ways to implement a Petri net model for this scenario. The reader can also imagine a single "operates under the assumption of clear weather" another single "operates under the assumption of inclement weather" place but with multiple colored tokens with transitions checking and only allowing certain colors. Simplifying the places usually results in complicating the tokens and transitions in Petri net models. Yet another alternative representation can create "weather information tokens" that are disseminated through the network reaching different nodes and multiplying along the way (they can then be destroyed by updated information that is disseminated later).

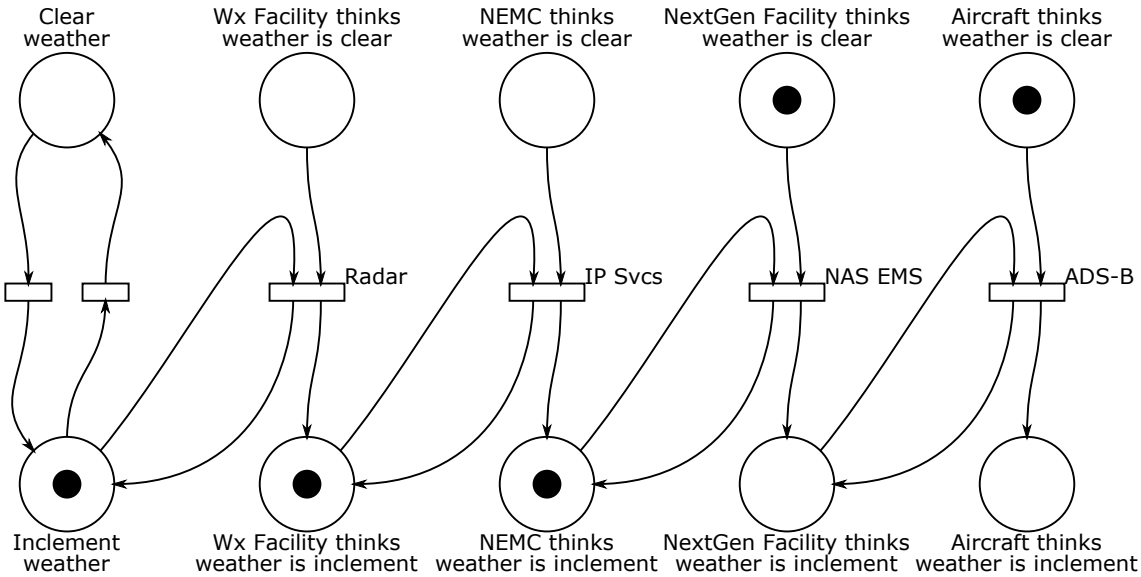


Figure 137: FAA Mid Term Petri net model for weather (partial)

All of the solutions mentioned do use the SV-6 on top of the SV-2. It is concluded that while the hypothesis stands (the architecture's views are sufficient to create effective Petri net models), the expectation of the SV-2 being enough alone is not

observed. SV-2 is still the basis of the Petri net model created, and cases in which it can be enough to create a model; however, as this example demonstrates, it is not guaranteed. The table for the SV-2 to Petri nets is updated with this knowledge gained from the experiment. The reader can find the final results in Appendix A.

8.3.3 NASEAF Far Term Architecture

The far-term architecture removes OV-5b, SV-5b, and the collection of SysML Block Definition Diagrams. The decision of removing the OV-5b must be discussed as the OV-5b has been the most important view for modeling purposes so far. This removal mainly affects the process models such as system dynamics, Markov chains, and discrete event models. Also, the OV-5b, SV-2, and SV-6 combination that existed in the near-term architecture is broken in this version. Nevertheless, the inclusion of SV-2 and SV-6 at least can help with defining the connectivity between systems in the models.

The removal of the BDDs also shows that the FAA is not perfectly clear about how to manage the National Airspace System this far into the future. The way the future systems communicate with each other is defined by the FAA regardless of how they may be internally put together. For these reasons, any models that are created from this architecture will be inferior to the ones created for the near-term architecture in detail. Given that these models are aiming to describe something that does not exist in the present, the lack of detail is perfectly reasonable.

Given the products created and the tables in Chapters 6 and 7, the following observations were derived. Table 198 shows the ordering of viewpoints in utility for each modeling type.

- Agent-based models cannot be created from a single viewpoint.
- The SV-4 is still the hierarchical type and is again ignored in this analysis.
- SV-2 and SV-6 are elevated to critical viewpoints for modeling.

Table 198: Observations from the far-term version of NASEAF

Model	Feasible	Views ordered by utility	Relevant Obs.
Graph	✓	OV-2 = OV-3 = SV-1 = SV-6, OV-1 = SV-2	2 and 3
Probability	✓	SV-2 = SV-6, OV-1, OV-2 = OV-3	2 and 3
System dyn.	✓	SV-2, SV-6, OV-1 = SV-1	3
Markov chain	✓	OV-1 = SV-2 = SV-6	3
Petri net	✓	SV-2, OV-1 = SV-4, SV-6	2 and 3
Queueing	✓	OV-2, SV-6, OV-1	2 and 3
Disc. event	✓	SV-6, SV-2, OV-1	2 and 3
Agent-based	∼	SV-1 = SV-2, SV-6, OV-1	all

Observations 1 and 2 have been discussed before for as-is and near-term architectures so these discussions will not be repeated here. Observation 3 means that processes will be difficult to model for the far-term architecture. While the process models rely on the SV-2 and SV-6, without the OV-5b, OV-6b,c, SV-10b,c they lack the main subject of what they are to model. The tables show that these models can still be possible to build. The tests that follow will yield a more definite answer.

It has been discussed that Type B experiments yield the most information earlier. However, in this case the only possible Type B experiment is the one with the agent-based model with SV-1, SV-2, and SV-6. The same combination was investigated in the near-term architecture and it would be a redundant repetition. Therefore, only Type C experiments are performed in this section: a probability model and a discrete event model.

8.3.3.1 Building a probability (Type C experiment)

The approach used here to construct a probability model assumes communications failure probabilities are known or estimated ahead of time. While simplistic, the approach is appropriate because the architecture is for a future design of the NAS; therefore, any probabilistic study would not operate on historical data to figure out the probabilities. Table 198 suggests the usage of the SV-2 and SV-6 for creating the

probability model. The SV-2's flowlines are assumed to be probabilistic as well as one of the systems (NAS Boundary Protection). The rest of the graphic is assumed to have perfect success rate (i.e., probability of one). Table 199 summarizes the assumed probabilities for various networks depicted on the SV-2.

Table 199: Probabilities used in the model from the FAA Far Term Architecture

Connection	Symbol	Assumed prob.
External DTS/Network Connection	α	0.99
Secure NAS Enterprise IT Infrastructure	β	0.999
DTS/Access Network	γ	0.99
Between Remote Facility and Aircraft	δ	0.95
Between Remote Facility and Satellite	ϵ	0.9
Between Satellite and External Nodes	ζ	0.92
NAS Boundary Protection (Node)	η	0.9

In addition to the SV-2 information, the rows from the SV-6 were used to obtain a more realistic modeling of the data communications. The SV-6 includes the data elements that are shared between systems and using the connections from the SV-2 the probability of each data element being communicated successfully can be calculated with a few lookup tables. Figure 138 shows how the information from the two distinct views are integrated together.

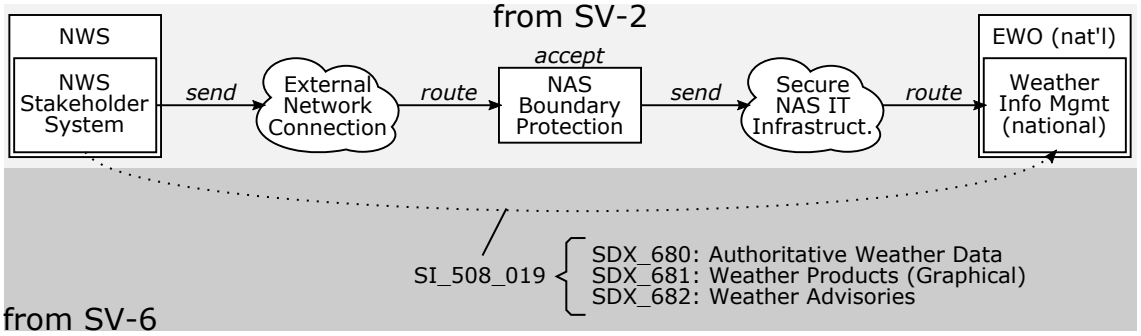


Figure 138: How parts of the SV-2 and SV-6 are used to create a probability model

Considering the connection between two systems depicted in the SV-2 and SV-6, we can calculate the probability of the message getting through by successive multiplications of the probabilities. For the example shown in Figure 138, this equation

becomes $P(S) = P(\alpha)\,P(\alpha)\,P(\eta)\,P(\beta)\,P(\beta) = 0.88$. Based on the same assumptions, Table 200 shows the probabilities for some of the other connections given in the SV-6.

Table 200: Calculated communications probabilities in the FAA Far Term model. Disclaimer: the numbers are based on assumed probabilities.

System Interface Identifier	Data Element Identifier	Prob.
SI_001_004	SDX_001	99.80%
SI_003_004	SDX_128	99.80%
SI_004_507	SDX_231	88.03%
SI_005A_504	SDX_251	88.03%
SI_005B_505	SDX_269	88.03%
SI_008_016	SDX_322	99.80%
SI_009_004	SDX_338	99.80%
SI_011_007	SDX_418	99.80%
SI_017_003	SDX_490	99.80%
SI_302_003	SDX_611	92.92%
SI_511_019	SDX_691	88.03%

8.3.3.2 Building an discrete event model (Type C experiment)

Based on Table 198, the SV-6 was the first architecture model that was used. It provides many of the required bits of information such as the events as well as the servers that are required for the events to execute; however, it lacks the information on the order of events, i.e., which events lead to which events. It also lacks details on what happens when events fail. Figure 139 shows how the information from the SV-6 flows into the discrete event model's input file. The reader can see that most of the basic structure of a model is included in the SV-6. The meaning of the arrows are explained below.

Arrow 1 The data element identifier is conveniently used as a unique name for *events*

Arrows 2 and 3 The events appear inside the process sequence

Arrows 4, 5, 6, and 7 The systems responsible for creating and sending the data are used as *servers*

Arrows 8 and 9 Recipient systems are checked for what events they perform in turn. These events become the subsequent events.

System Interface Identifier	Data Element Identifier	Data Element Name	From Containing System Node	From System Entity	To Containing System Node	To System Entity
SI_004_017	SDX_216	Flow Management Constraints	Enterprise Wide Operations (National)	Flow Contingency Management	Enterprise Wide Operations (National)	Trajectory Management - National
SI_004_017	SDX_217	Sequencing Plans	Enterprise Wide Operations (National)	Flow Contingency Management	Enterprise Wide Operations (National)	Trajectory Management - National
SI_004_017	SDX_218	Flight Assignment to Flow - FCM	Enterprise Wide Operations (National)	Flow Contingency Management	Enterprise Wide Operations (National)	Trajectory Management - National
SI_004_502	SDX_219	Flow Management Plans	Enterprise Wide Operations (National)	Flow Contingency Management	Airport	Airport Stakeholder System
SI_004_502	SDX_220	Flow Management Analysis Results	Enterprise Wide Operations (National)	Flow Contingency Management	Airport	Airport Stakeholder System
SI_004_502	SDX_221	Flow Management Plan Inputs	Enterprise Wide Operations (National)	Flow Contingency Management	Airport	Airport Stakeholder System

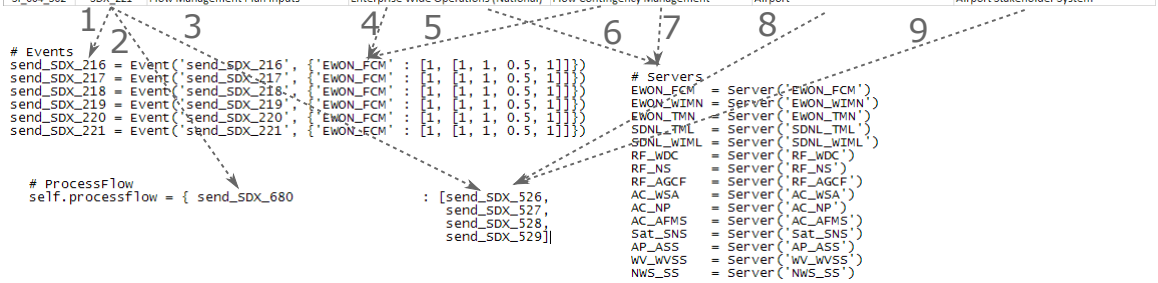


Figure 139: Translating the FAA Far Term SV-6 to a discrete event model

While the majority of the basic structure is there, the details on where the whole process starts and ends or which events actually cause which events are not included in the SV-6. With the above method, once a data element reaches a node, it sends out all possible data elements it can. This may be not realistic for some problems. For problems that need a better sequencing, an SV-10c or OV-6c may be required. Nevertheless, the model can be built and can be executed practically. Figure 140 shows a selected output.

The Far Term architecture also includes an SV-2 that includes information on the middle steps for the communication links between two systems. These routing and switching between networks can also be modeled as events with different servers. However the events from the communications will be happening at a much higher rate than the others and it may make more sense to create an entirely separate discrete event model from the information obtained from the SV-2. In any case, it appears to be entirely plausible. In fact, Table 198 suggests the SV-2 as the next best view after the SV-6.

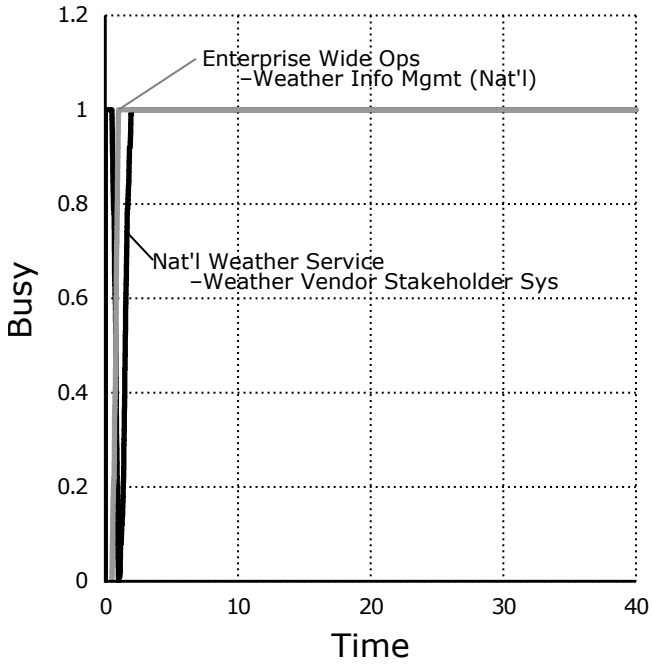


Figure 140: Translating the FAA Far Term SV-6 to a discrete event model

8.4 Discussion on the Research Arguments

The purpose of this chapter was to test the element maps to use them in support for the two research arguments. The tests were done through the architecture elements to modeling element maps constructed in Chapters 6 and 7. The element maps allow for the testing of multiple arguments in a merged fashion. If the maps are valid, Argument 1 holds: architectures are conceptual models in the context of simulation modeling. Additionally, the maps can be used to find all alternative simulation models for a given system of systems. This is then used to test Argument 2. This discussion continues in the next chapter.

The maps hold well against the tests performed in this chapter, which allows for the use of the maps for the support of the arguments. Apart from a single hiccup, the tables appear to work as intended. These tables that are scattered throughout Chapters 6 and 7 are combined into four larger tables and given in Appendix A. In the next chapter, the maps will be used to argue for both of the research arguments.

CHAPTER IX

CONCLUSIONS AND THE SOLSTYSS METHODOLOGY

In Chapters 6–8 the architecture and modeling element maps were created and tested. Having justified the element maps, it is put to use to support the research arguments presented in Chapter 3. Additionally, the map will be used to identify best modeling fit for each viewpoint and vice versa. A few examples for combinations will also be provided. Finally, the methodology of using this map will be presented so that the work can be useful in future technical work.

The tables developed in Chapters 6 and 7 are compiled into a set of tables where studying each modeling type's ability to represent an architectural element is significantly more convenient. These tables are one of the major contributions of this work and can be used for future system of systems design and analysis problems outside of this work. The compiled tables are quite large and are presented in Appendix A.

9.1 Support for the first research argument

As can be seen from the element maps in Tables 204–207 the architecture views, when broken down into elemental parts, can be *translated* into modeling elements. When these modeling elements are compiled into a computer code, the model becomes executable. In all of the examples discussed in previous chapters, architectures could be turned into executable models. Figure 141 shows this process of translating an architecture to executable models.

The argument (architectures can be translated into conceptual models) is then supported through the process outlined in Figure 141. Architectures are a collection of architecture viewpoints that are made out of architecture elements that can be translated into modeling elements that make up a conceptual model. The conceptual

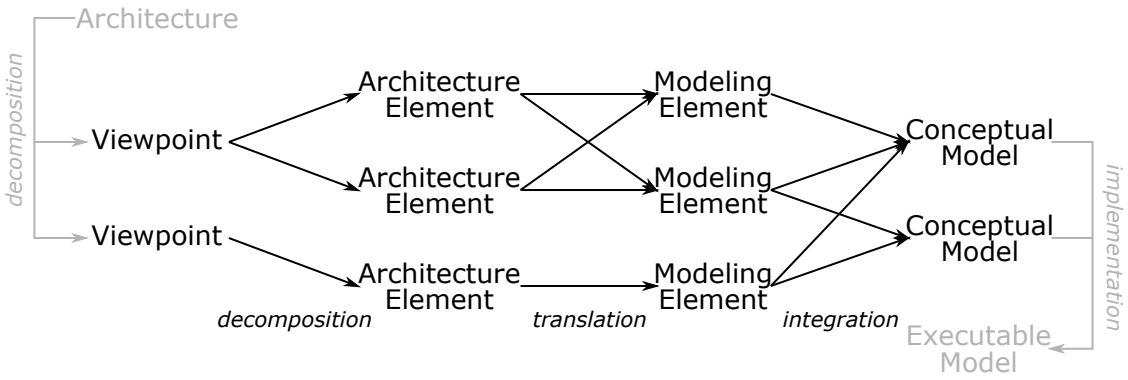


Figure 141: How architectures become models

model can be implemented in a code that can be executed on a computer. This process showcases how architectures are equivalent to conceptual models for the purposes of computer modeling. If a modeler has an architecture instead of a conceptual model, a computer model of the system of systems can still be built.

While architectures are shown to be convertible to conceptual models, there is no guarantee that any architecture can be translated into any conceptual model or executable computer simulation. For example, in Section 8.2.1.1 the agent-based modeling had failed. Not every system of systems architecture can be expected to have enough structure to support all possible kinds of modeling. Architectures may be missing details, may not be consistent, or may not be complete descriptions of what a system of systems is composed of or how it operates. In fact, if an architecture cannot be turned into a desired executable model, it is a possibility that the architecture is not a complete description of the system of systems and must be expanded.

Additionally, an architecture complete enough to create a type of model must not be expected to be necessarily complete enough to create another type. This is especially apparent when comparing one conceptual model for a computer model to another conceptual model for another computer model. For example, a Markov chain transition matrix must not be eliminated as a conceptual model just because it cannot be used to make discrete event models. We cannot apply such a rule on

architectures either. Therefore, the fact that not all system of systems architectures can be translated into every modeling type does disprove the research argument that system of systems architectures are conceptual models. As long as architectures can be translated into one conceptual model, the argument holds. Every element of every modeling type investigated before was obtainable from one or more architecture elements.

It is also important to note that just like conceptual models, architectures are not by themselves executable. Execution ability requires a very specific implementation of the model in a scripting or programming language. While the selection of language is not the most important decision in the process of creating the executable model, it is a necessary step. Therefore, an architecture's computer model may look completely different based on mechanical decisions made by the computer programmer. There is no one-to-one map, whereas for a specific conceptual model and the architecture it comes from, there is a strictly one-to-one map. Once the modeling type and context are set, the conceptual model is unique, while the computer model is not.

In conclusion, the first research argument is supported through the element maps reading from rows to columns. The first(decomposition), second(decomposition), and fourth(integration) steps depicted in Figure 141 are trivial given Tables 1–3 and the fifth step is not critical for arguing for possibility, it is only a practical necessity and implementation decision. However, the third step was not obvious at the start of this work. The element map constructed in Chapters 6 and 7 and subsequently tested in Chapter 8 provides a template to achieve the translation necessary for a given system of systems problem. With the existence of this map, research argument is deemed to be valid.

9.2 Support for the second research argument

The second research argument can also be supported by the element map table. Here the investigation is on the claim that multiple modeling types are required to model practical systems of systems, i.e., there is no single modeling type that can cover all aspects of system of systems. To test this, the examples in Chapter 8 are investigated first.

The RWDC example is studied first. Jones Wyatt developed three views for it: SV-1, SV-6, and SV-7. These three views combined provided her with enough information to build probability models to study the effects of the interoperability between systems. However, none of these views depicts the operations done by each of the systems. The viewpoints developed are entirely structural and limited to data communications.

While her model as well as the models tried in Sections 8.2.1.1, 8.2.1.2, and 8.2.1.3 were aimed at simulating the system of systems in a single model, it can be easily seen that the description of the system of systems is not complete. In fact, UAVs flying around and searching for lost hikers—the original description of the system of systems' operation—would almost certainly benefit from system behavior descriptions such as what the UAVs do when they get too close to each other or when manned aircraft are introduced into the airspace, e.g., rescue helicopters. Appropriately, such a model would be inadequate in measuring the metrics that Jones Wyatt was interested in. This example does not appear to be the counterexample this hypothesis testing required. The research argument stands.

The second example is the FAA NASEAF's as-is version. This architecture relies only on operational viewpoints: OV-1, 2, 5b, and 6c. A similar problem to the previous problem surfaces here. These viewpoints do not describe the system of systems fully. Without full descriptions of the systems and their behaviors, it is difficult to assess the performance of this system of systems completely. Section 8.3.1.1 describes

how two modeling types can be used together to cross-check results. Additionally, the OV-5b used in the modeling activity does not depict the overall network structure and this structure is not studied in the discrete event model. The resource network is given in OV-2. This example is not the counterexample needed to invalidate the hypothesis either. The argument stands.

The third example is the FAA NASEAF's near-term version. This architecture is one of the most complete architectures studied here. The most complete model that was built was the agent-based model. This model used a significant amount of information from multiple views, as detailed in Table 197. It can be readily observed that operational requirements in data are not studied with this model. The agent-based formulation that is based on systems does not provide a good understanding of needs of each operational nodes. To study such a network of needs, a graph or system dynamics model would be more appropriate.

Additionally, there was not enough detail to describe systems in the model so that their behavior would match the real systems. Rules were mostly missing, how to deal with each data piece was also missing. All-in-all the agent-based formulation was a good attempt at modeling this system of systems even though it did not provide the necessary metrics to study all aspects of this system of systems. The hypothesis still stands.

The same trend repeats in the far-term version. Discrete event is a mixed success. The structure appears to be there; however, the rules of what to do with each data item were missing. Additionally, an alternative network for these data connections would have been difficult to investigate with the discrete event model. The probability model that was constructed from other views included in the architecture deals with such a study in a much easier, more succinct, and more dependable way. It can also perform similar studies on alternative networks quickly. The hypothesis therefore still stands.

Another approach to falsify the hypothesis is taken in the construction of Table 201. Using the element maps and reading from columns to rows, each model's inability to model various architecture elements are identified. If a row under a specific modeling type has entirely *No* marks, that element cannot be modeled well using this specific modeling type. As expected and observed in the table, no modeling type has full coverage when it comes to modeling all aspects of systems of systems.

Table 201: Missing system of systems aspects in each modeling type

Model	What is missing?
Graph	Rules that apply to operations and systems (e.g., OV-6a), timeline information related to operations and system functions (e.g., OV-6b,c and SV-6a,c)
Probability	A large number of system of systems aspects cannot be modeled using probability models. Examples include organizational structures, timelines, rules, and taxonomies.
System dynamics	Taxonomy, rule based activities, timeline-based discrete transfers, function and activity allocations, and future planning
Markov chain	Taxonomy, rule-based activities, time-based activities, implementations of resource transfers, function and activity allocations, and forecasts
Petri net	Organization, taxonomy, interaction-based operations and functions, allocations, and forecasts
Queueing	Operational resource flow, organizational structure, taxonomy, rule-based activities, allocations, system rules, and condition-based activities
Discrete event	Organizational structure, taxonomy, operational rules, some allocations, and planning forecasts
Agent-based	Operational needs, taxonomy, operational rules, and technology forecasts

In light of the support given above, the second research argument is deemed to be valid. It is recommended that system of systems engineers, designers, and operators build multiple types of models to:

- increase the coverage of system of systems aspects that are being modeled,
- cross-validate models against each other as different types necessitate different

ways of thinking about the problem, and

- some metrics are easier to measure using different modeling types which reduces time to model.

9.3 Best modeling types for each viewpoint

Using the element maps, a best modeling type for each viewpoint map can be constructed. To do this, architecture elements associated with a single viewpoint are isolated and checked against all possible modeling elements. If each architecture element maps to a unique modeling element and all modeling elements are obtainable, it indicates that this viewpoint is very appropriate to create models of this type. If such a map does not exist, then the most complete map is sought, i.e., all modeling elements are covered but the map is not one-to-one. Table 202 shows the results of this investigation. Additionally, the reader can investigate tables created in Chapters 6 and 7.

Some notable patterns emerge from the Table 202. Graph models appear to be the best modeling types for the earlier, less strict, less detailed viewpoints. The result is not unexpected, graph models can be built with very small amounts of information and can provide rudimentary analysis capabilities easily. More detailed and technical viewpoints appear to favor process-based modeling types such as Petri net and discrete event models. System dynamics models appear to be a convenient middle-ground for a transition between lower-fidelity models to the higher-fidelity models. The ability to model processes at the highest level gives system dynamics a significant advantage over other models for this role. The author recommends system dynamics models for the development of virtually all systems of systems.

Studying Table 202 also uncovers that probability models, Markov chains, and agent-based models are never the best modeling types for any of the viewpoints. There are several reasons for this observation. Probability models suffers from the

Table 202: Best modeling types for each architecture viewpoint

View	Graph	Prob	SD	MC	PN	QT	DES	ABM
OV-1	✓							
OV-2	✓							
OV-3	✓							
OV-4	✓							
OV-5a								
OV-5b			✓					
OV-6a					∼			
OV-6b			✓		✓		✓	
OV-6c							✓	
SV-1	✓							
SV-2			✓		✓			
SV-3	✓							
SV-4						✓		
SV-5a	∼							
SV-5b	∼							
SV-6	✓							
SV-7								
SV-8								
SV-9								
SV-10a					∼			
SV-10b					✓			
SV-10c						✓	✓	

fact that graph models are very similar to them but more general. In some cases probability models will be more appropriate to use rather than graph models; however, this is difficult to predict ahead of time just by knowing a specific viewpoint exists in the architecture. If the viewpoint describes a stochastic process, then probability models would be more appropriate, but if the process is very mechanical and fairly deterministic, they would have significantly less utility. It is impossible to know ahead of time. Therefore, graph models are deemed to be a safer and more generally useful option when modeling with these earlier viewpoints.

Markov chains suffer from the fact that they are very appropriate for modeling the state of a single system can be in. The number of states grow combinatorially and is in most cases impractical. There are possibilities in translating viewpoints into Markov

chains using other views for scoping states but that is not the topic investigated here. This discussion is based on a single viewpoint. Appropriately, agent-based modeling suffers from the lack of a single view that can explain each system's behaviors and interactions with the rest of the systems as well as the environment. For both of these modeling types the impossibility is due to one viewpoint not being enough while the details of the problem being entirely different.

9.4 Best viewpoint to develop for each modeling type

Similar to the previous section the element maps are used to find the best viewpoint to develop for each modeling type. This analysis requires the reading of the map starting from columns and going towards rows. Naturally, some of the models identified to be best alternative for modeling for each viewpoint have a return relationship: the viewpoint is the best viewpoint to develop for this modeling type. In order to construct this table, a group of columns associated with a modeling type are mapped to the group of columns for each viewpoint.

If there are one-to-one relationships, the viewpoint is taken as the best viewpoint for that modeling type. There may be more than one such viewpoint. Alternatively, if there is no such one-to-one map, then a complete coverage is sought: all modeling elements are catered for. If no such map exists either, the most complete map wins. The results are given in Table 203. The reader can also investigate the tables in Chapters 6 and 7 for further details.

As expected, many viewpoints provide significant amounts of information for building graph models. Probability models struggle again because they require the viewpoints to depict stochastic processes. SV-10b appears to be a good choice for probability models if it depicts non-deterministic processes. For system dynamics, process and flow viewpoints appear to win as expected. System view flow models work better with system dynamics because the flows can be measured and actually

Table 203: Best viewpoint for each modeling type

View	Graph	Prob	SD	MC	PN	QT	DES	ABM
OV-1								
OV-2	✓							
OV-3	✓							
OV-4	✓							
OV-5a								
OV-5b			✓					
OV-6a								
OV-6b			✓	∼	✓		✓	
OV-6c								
SV-1								
SV-2			✓	∼	✓			
SV-3	✓			✓				
SV-4						✓		
SV-5a								
SV-5b								
SV-6	✓			✓				
SV-7								
SV-8								
SV-9								
SV-10a								
SV-10b	✓	∼						
SV-10c						✓	✓	✓

physically implemented. Markov chains and Petri nets prefer similar viewpoints as well although the *best* viewpoints do not match exactly. Queueing and discrete event models prefer process as well as timeline viewpoints because queueing and discrete event models rely on timing information greatly. Finally, agent-based does not have a view that can provide all the information necessary; however, the SV-10c appears to have the most complete information among all other viewpoints.

Table 203 has a few interesting patterns worthwhile of discussion. One such pattern is the use of SV-10c for the very detailed modeling methods such as discrete event and agent-based. It is apparent that the SV-10c is the main view for higher fidelity modeling and must be developed if such models are required during the system of systems design. SV-10c provides the much needed information on timing, interaction

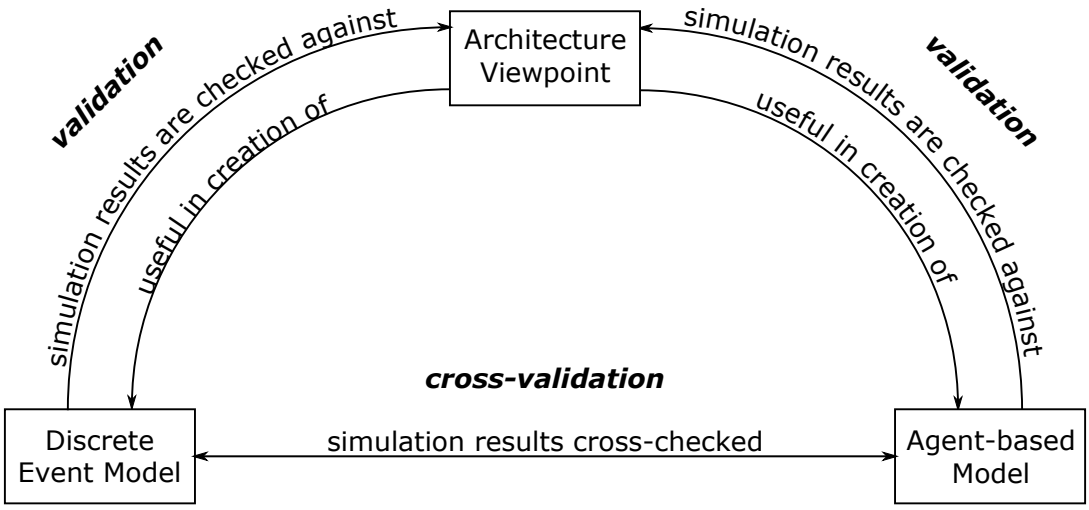


Figure 142: Architecture views that are instrumental in creation of multiple modeling

between systems, internal processes of systems.

The SV-3 and SV-6 fit well together from the point of modeling. Models that can benefit from SV-3 heavily, can do the same from the SV-6. A similar pattern emerges between the OV-10b and SV-2; however, these viewpoints have no similarities as architectural descriptions. OV-10b depicts how operations transition from one state to another while the SV-2 details the way resources are shared between systems. Putting them together can enhance the fidelity of flow and transition models by mapping which resource flow is needed at which state of the operation.

Finally, the utility overlap of one viewpoint with two or more modeling types hits towards cross-validation and hybrid-modeling opportunities. For example, the development of an SV-10c helps with building discrete event as well as agent-based models and enables the cross-validation between them. Regardless of the model's logic and mechanics, both models must agree with the SV-10c's timeline depiction. Once both models agree with the SV-10c, i.e., they are validated with the view, other outputs of the models that are not described in the SV-10c can be cross-validated with each other and fixed until they converge. Figure 142 depicts this process.

Another pattern emerges from Table 203: some views, while useful for modeling,

are never the best views for creating models. OV-1, OV-6c, and SV-1 are all useful in modeling systems of systems but they never appear as the *best* view to produce for modeling systems of systems in any type of modeling. It is recommended that these views are generated as secondary views supporting other views in the effort for modeling.

9.5 Recommended work flow for modeling systems of systems using their architectures

The main recommendation from this work is to explore opportunities for building multiple models of various types when analyzing systems of systems. It has been shown in several studies that singular modeling approaches to system of systems engineering problems will leave some aspects unmodeled. These studies were all based on the element maps conveniently summarized in Appendix A. System of systems engineers are recommended to look into these maps and look for opportunities in creating alternative models with the given set of architecture views they currently possess. Roughly the idea is to find alternative models that can be supported by the existing architecture views.

- the rows in which the existing views are listed are isolated and other rows removed
- with the remaining rows, each modeling type is investigated individually
- if enough coverage of modeling elements exists within the isolated rows, that modeling type can be tried

The second recommendation is to always look for cross-validation opportunities. Using the same element maps, overlaps in modeling element to architecture elements can be identified and multiple models can be trained against each other until both converge on the same results in shared metrics. Using such cross-validated modeling

approaches the list of metrics analyzed and investigated can be expanded with increased confidence in their correctness. Roughly, the steps to identify cross-validation opportunities are as follows.

1. identify columns belonging to the planned modeling type
2. identify rows belonging to the architecture views that exist for this system of systems as well as used for this modeling type
3. look into other columns and identify other modeling types that can be built by these rows
4. find rows that have *Yes* entries under a modeling element in these new modeling types that also have a *Yes* entry under the original modeling type
5. determine whether that new modeling type can be built by the views that exist for system of systems in general

The third process is to identify missing information necessary to perform higher-fidelity modeling. At any point in the system of systems development process with a given set of architecture views, it may not be possible to create some higher-fidelity models. Using the element maps, the reader can identify which architecture views are missing to create the next executable model that can analyze the system of systems in a more detailed way compared to the previously tried modeling types. The procedure is outlined below.

1. pick a higher-fidelity modeling type such as discrete event or agent-based, remove all other columns
2. highlight architecture views readily available
3. looking in the columns identify modeling elements not covered by the available architecture views

4. identify other architecture views that can add the missing information

5. make a decision on which one to add:

    - select the minimum number of new views
    - select the views that are easier to develop based on prior knowledge about the system of systems
    - select the views that are also useful in performing other analysis activities in the system of systems engineering effort

6. develop the view(s) and develop the executable model for analysis

System of systems engineers can benefit from applying these processes using the element maps on their problems. In order to systematize these processes and include other modeling decisions that can be made outside of the context of this work a methodology named Selection Of Logical Simulation Types for Systems of Systems (SOLSTySS) is presented here. The methodology has two main elements: a flowchart and the element maps it heavily uses. The flowcharts are constructed to describe the process of making model type selection decisions and are given in Figures 144–148. The chart references the element maps (Tables 204–207) as well as the best modeling type for a given view table (Table 202). Due to the size of the flowchart (Figure 143), it is broken into 5 main parts. System of systems engineers can start with Part A (Figure 144) and follow the exit and entry points to the other parts (Figures 145–148).

In the flowcharts, enter and exit nodes are drawn as circles, decision nodes as diamonds, and actions as rectangles. Some rectangles require the construction of lists or numbers as a function of architecture views and modeling types in consideration and will be discussed. When "Develop model" nodes are reached, a model-building exercise is necessary. Examples of model building exercises were given in Chapter 8. Usage of element maps are recommended during the model development as well as during the selection.

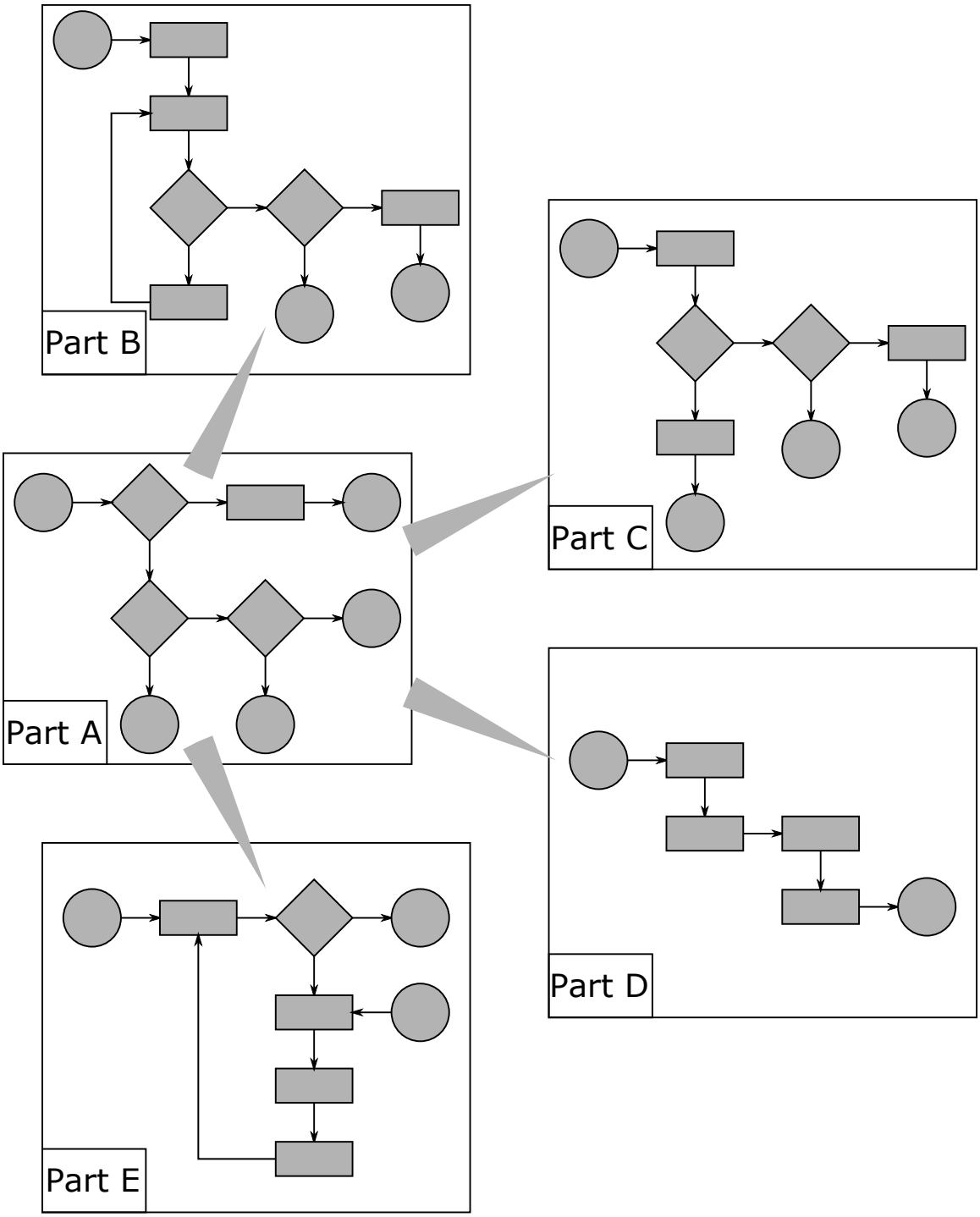


Figure 143: SOLSTySS flowchart has 5 different parts. Part A is the starting point and is always executed. Other parts are executed if they fit the scenario.

The flowcharts are constructed to help specifically with the selection of model which is the topic of this work. Once the system of systems engineers select a model and develop it, it does not mean that they must stop. At any point the system of systems engineer can restart the process to pick alternative models for reasons such as seeking cross-validation opportunities, or developing higher-fidelity models. Some scenarios will be discussed next.

There are 4 main different scenarios for developing modeling within the SOLSTySS methodology. The first one is when there are no previous models built for this system of systems. This case corresponds to a very early design scenario. In order to get to this case Figure 144 is used. In the conditional nodes, A1 is selected to be false and A3 is also selected to be false. This scenario leads to Part B given in Figure 145.

The first rectangle (B1) references a set denoted by the symbol $V^{\star}$. This is the set of all existing architecture views for the system of systems. Throughout the development V will be used as a list of all architecture views and any set modified with the $\star$ sign will denote the subset of the set with already existing members.

The second rectangle constructs a list denoted by Λ_1. This list includes all modeling types that are determined to be the best modeling types for to the most detailed existing view. The determination is based on Table 202 which is denoted as B_M. The set is the set of all models that have a checkmark on the row specific for the most detailed view in the existing views. Λ_1 is defined in Equation 74.

$$\Lambda_1 = \{m \in M \mid B_M(v, m) = 1\} \tag{74}$$

If the row is empty (B3 is false), then the system of systems engineer can pick the next most detailed view and continue the process. There is a danger of running out of views here although it is extremely unlikely. If this problem occurs, models must be selected by different criteria (A1 is true).

If there are more than one such "best" modeling types (B5 is true), the modeler is given a freedom to choose using some other criteria. If there is only a single modeling

type (B5 is false), then that type is selected. Once a modeling type is selected, the process moves to Part E which is shared by multiple scenarios and is shown in Figure 148. The next box is E1.

The process in E1 requires the construction of another list denoted as Λ_2. This list includes all modeling elements specific to the selected modeling type, i.e., columns of the element maps, that do not get supported by the already existing architecture views. To construct this list, each column corresponding to a modeling element for the chosen modeling type is checked against all the rows that correspond to existing views. If there are no "Yes" entries in the column for these rows, then that modeling element unsupported and is added to the list.

Equation 75 details the operation succinctly. C^m denotes the columns of the element map specific to the selected model, $R^{V^\star}$ denotes the rows of the element maps specific to the existing views combined. Finally, $\#_Y$ is a function that counts the number of "Yes" entries in a given list of things.

$$\Lambda_2 = \left\{ c \in C^m \mid \#_Y \left[E\left(R^{V^\star}, c \right) \right] = 0 \right\} \tag{75}$$

If all modeling elements are supported, i.e., $\Lambda_2 = \emptyset$, then E2 is true and the next step is to develop this selected model and the model selection process is terminated. If however, Λ_2 is not empty, then new architecture views must be developed to support model-building. The rectangle E3 requires the calculation of a number for each view that does not exist in the architecture.

For each view the number of "Yes" entries in the element map for each modeling element lacking support is counted. If the count is larger than 0, this view's support for that modeling element is adequate. Then the adequacy is summed across all modeling elements needing support. Equation 76 shows the calculation. This number, S_v, provides the utility of the architecture view in filling the gap for the modeling that was selected. The view with a maximum S_v is then selected to be developed and the process returns to E1 for the determination of whether the modeling type

selected can be supported now.

$$S_v = \sum_{m \in \Lambda_2} \sum_{i \in R^v} \{\#_Y [E(i, C^m)] > 0\} \tag{76}$$

These steps conclude this scenario. The next scenario to be investigated is when a model already exists but a cross-validation model is needed to improve confidence in the existing model. In this scenario, starting in Part A (Figure 144), A1 is false, A3 is true, and A4 is also true. These conditions lead to Part D given in Figure 147.

The first step is a trivial definition step. The model that needs to be cross-validated is defined as m_c. From this a list named Λ_3 is constructed. This list contains all the alternative models that can be constructed from the same views used to construct m_c and is defined in Equation 77. The mechanics of this selection find rows in the element maps (architecture elements) that map to columns (modeling elements) corresponding to both models.

$$\Lambda_3 = \left\{ m \in (M \setminus m_c) \mid \#_Y \left[E\left(R^{V^{m_c}}, i\right) > 0 \right] \forall i \in C^m \right\} \tag{77}$$

Next a number named S_m for each of the models in Λ_3 is calculated using Equation 78. This number measures the number of rows where hits to columns under both models are found. The idea is to find as much overlap as possible; however, maximization is not necessary. A model with a large S_m should be selected for cross-validation exercises and developed.

$$S_v = \sum_{r \in R^{v^{m_c}}} \{[\#_Y (E(r, C^{m_c})) > 0] \wedge [\#_Y (E(r, C^m)) > 0]\} \text{ , where } m \in \Lambda_3 \tag{78}$$

The final scenario to be investigated is when a model already exists but other models are desired to simulate other aspects of the system of systems or simply an alternative formulation is desired. In this scenario, starting in Part A (Figure 144), A1 is false, A3 is true, and A4 is false. This scenario leads to Part C given in Figure 146.

The first step is to construct a list of alternative models that can be constructed from the existing architecture views. This list is defined as Λ_4 and is given in Equation 79. The list is compiled from a subset of modeling types that have not been

made. The columns of the element map that correspond to each such model are checked against the rows of the element maps that correspond to existing architecture views. If there is at least a single "Yes" entry for every column associated with a model, that model remains in the list.

$$\Lambda_4 = \left\{ m \in (M \setminus m^\star) \mid \#_Y \left[E\left(R^{V^\star}, i\right) > 0 \right] \forall i \in C^m \right\} \tag{79}$$

There are 3 possible outcomes of this analysis: there are no possible alternative model types, there is exactly one possible alternative model type, and there are more than one possible alternative modeling type. If the answer is one, then that model is simply developed. If the answer is more than one, then other external criteria are used to select one among them and that model is developed. Finally, if there are no possible alternative model types, the process moves to E3 where one or more architecture views are identified to be developed before modeling can proceed.

These three scenarios conclude the main cases to select modeling types based on architectures. However, it must be noted that there are other reasons to develop one modeling type over others. One such reason is the types of metrics a model can calculate during simulation. As discussed in the early stages of the document, selecting model types based on metrics is necessitated by the desire to make decisions supported by such metrics.

While not the main focus or contribution of this work, this case is semi-supported. Starting with the Table 201 and including other considerations not covered by this work, a model may be selected for development. While this selection is not based entirely on the architectures, the SOLSTySS methodology can support a part of the process in Figures 144 and 148.

Starting with Part A, the first condition (A1) becomes true, and based on external criteria in A2 a specific modeling type is selected. The SOLSTySS methodology then helps with the determination of whether there are enough architecture views to support this type of modeling which is the main focus of Part E. If enough architecture

definition is available, the model can be simply developed. If there is not enough definition, then views that are most useful are identified in Part E and developed accordingly until the architecture definition is enough for the specific model type building to proceed.

Apart from the case discussed above, the first model may have been selected without considering architecture view coverage. Adding to this model, a cross-validation model can still be selected using architectures and following the Part A to Part D. This demonstrates that even if architectures are not to be used in the initial model selection process, the SOLSTySSmethodology presented still has a reduced but significant utility to the system of systems engineer.

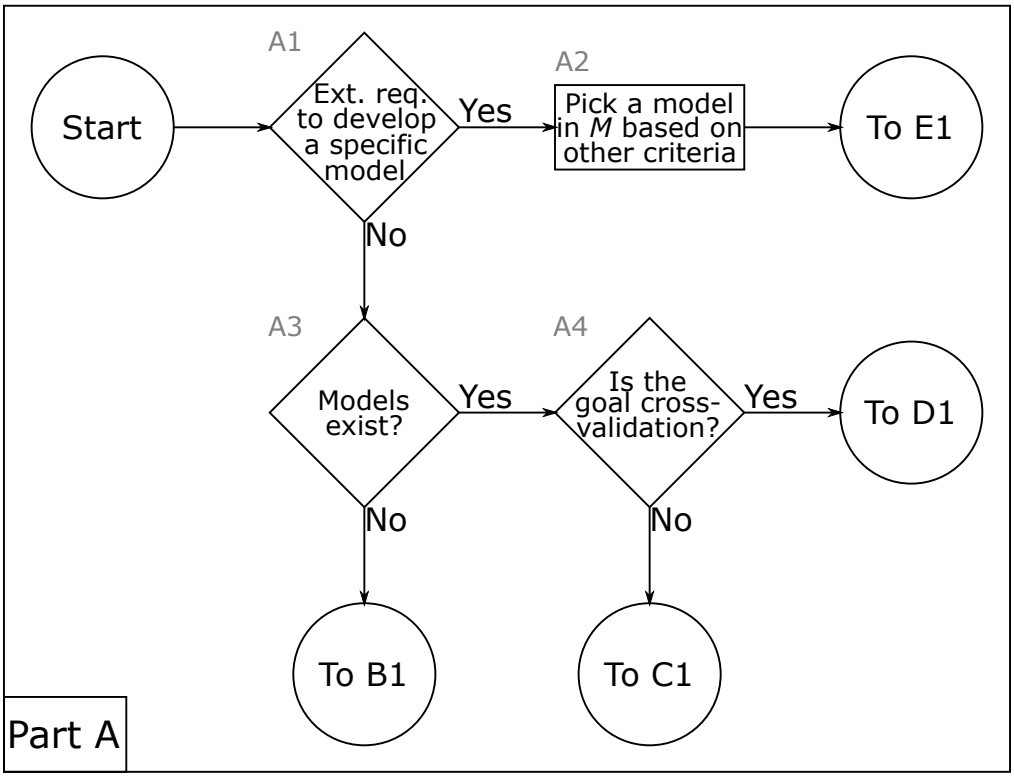


Figure 144: Starting point for the methodology. Scenarios are selected here.

The flowchart's purpose was to find which modeling elements to focus on but not do the modeling translation. Once a develop model node is reached, the next step is this translation of architecture elements to the modeling elements. The translation

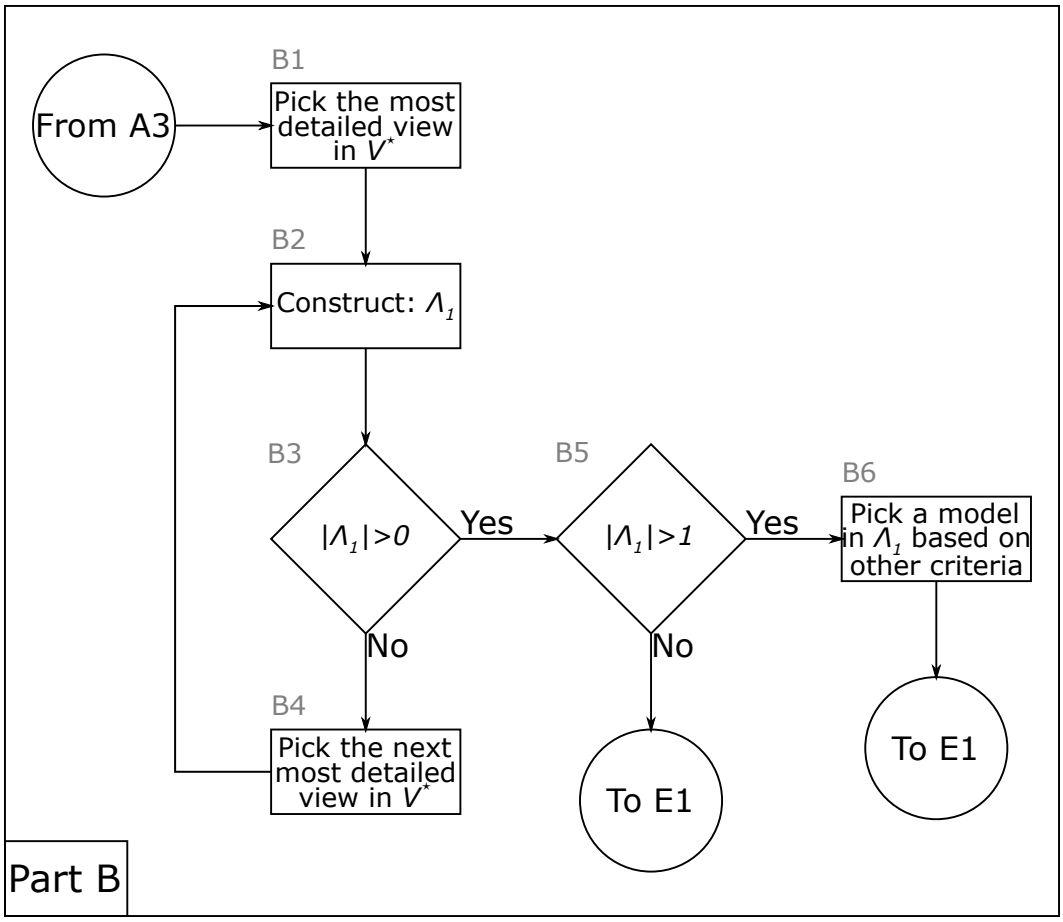


Figure 145: If no previous models exist, this part of the methodology is used.

is still a largely manual task. The element maps given in the Appendix A help with this actual translation step as shown in Figure 141. The reader is referred to the examples provided in Chapter 8 for the implementation of the translation for their problems. The element maps provide a convenient starting point of how information from the system of systems architecture is transfered to the conceptual model of the computer model. For specific language implementations, the conceptual models can be translated into code using discipline-specific processes. That final step should be straightforward for a practitioner.

A note about the evolving nature of systems of systems is necessary here. Most systems of systems change as new systems are introduced to the mix. When new systems are introduced as extra viewpoints but as existing types, there will be no

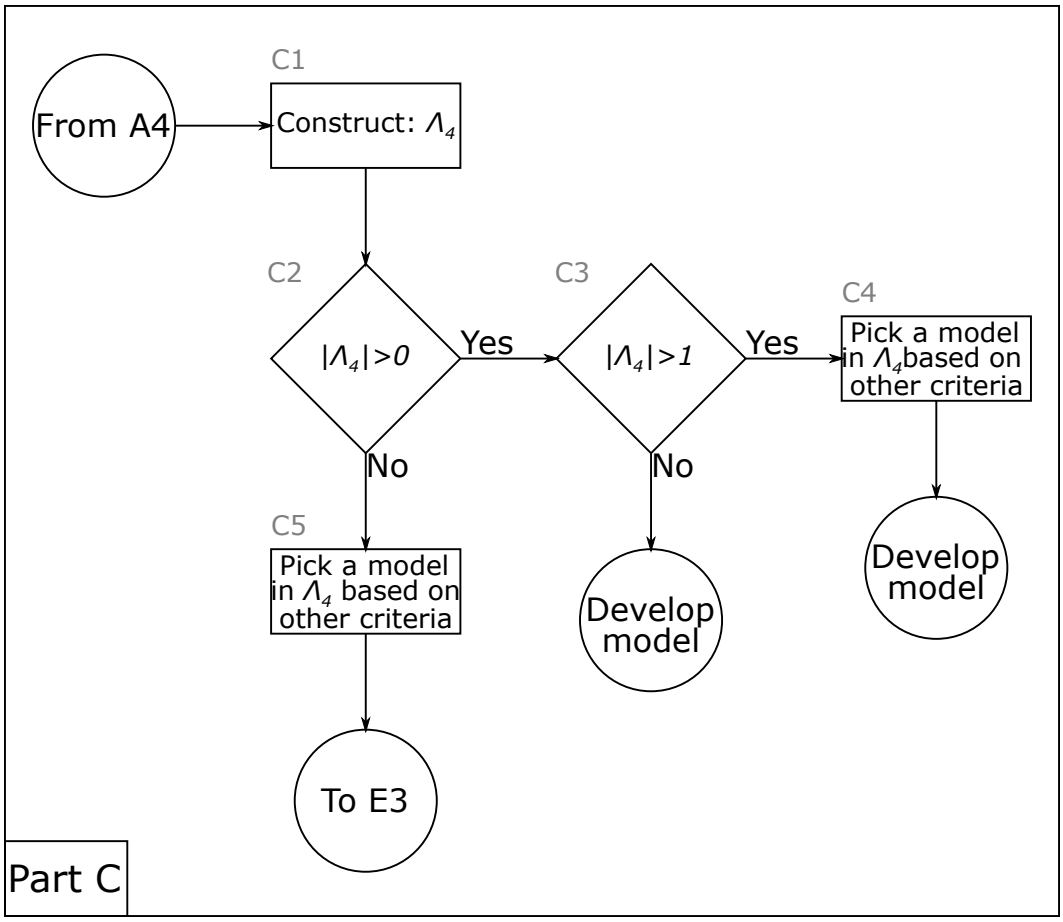


Figure 146: If some alternative models are sought with no particular requirements, this part of the methodology is used.

need to adjust the modeling type. Naturally, the new systems must be coded into the old models to see the effects of the new systems; however, the modeling type will not need changing. It is a similar problem with an extra system.

If a system or service's introduction causes new viewpoints types to be developed, then there is no guarantee that existing models will be adequate. In such scenarios, the methodology's Part C must be exercised with an extra attention to this new view by requiring it to be part of the solution.

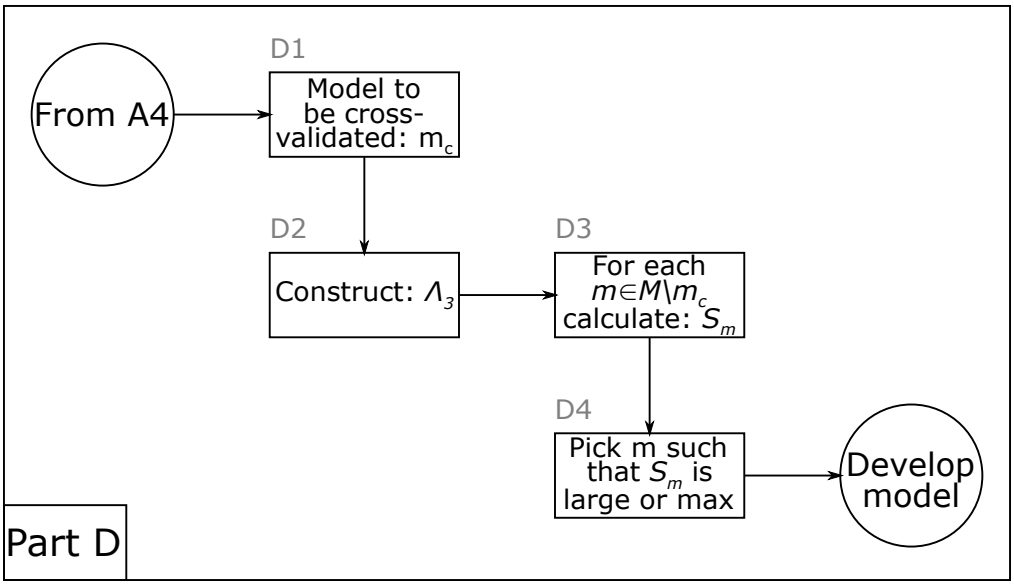


Figure 147: If a cross-validation model is desired for one of the existing models, this part of the methodology is used.

9.6 Future work

During the course of this work, several ideas for future work were generated without being aligned tightly with the goal of selecting modeling methods from system of systems architectures. This section is a compilation of such ideas that did not fit within the scope but will be investigated in the future as opportunities present themselves.

The first idea was to select models based on the metrics required to make certain decisions. As discussed in the early chapters, the metric-based selection is a perfectly reasonable approach to selecting simulation models. For example, in Section 4.3.2 one definition of a model was given by Minsky. In that definition, the questions a model needs to answer can include very specific metrics and numbers and other quantitative measurements. The goal and approach are slightly different to what has been detailed in this work which focuses on the replication of system behavior as described in the architectures. If the modeling approach is metric-based, then the work requires a pre-determined decision making problem. Based on that problem, required metrics can be determined, and finally models that can compute such metrics created. This

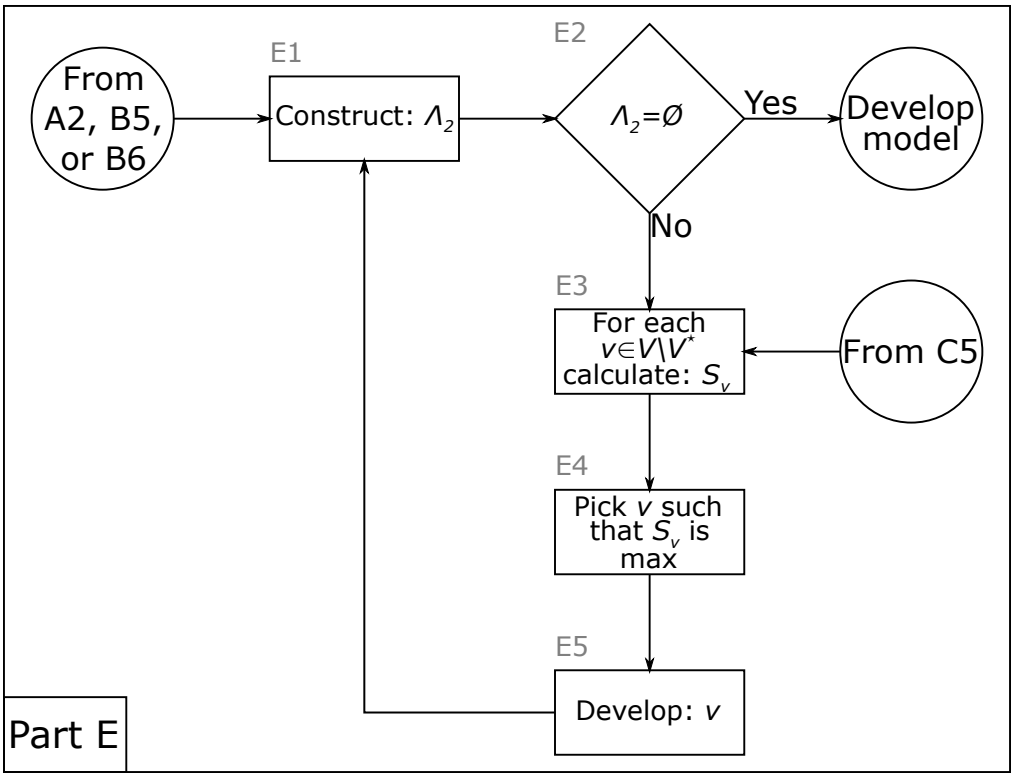


Figure 148: In some cases, especially if views are not enough to do a specific kind of modeling, this supplementary part of the methodology is used.

study would aim to help decision makers to get to the numbers to make decisions which is highly valuable while categorically different from the aim of this work.

Table 201 is a good starting point for studying metrics and their influence on model selection. However, as mentioned in Section 3.4, enumerating all aspects of systems of systems is not practical. In this work, architectures were used as a proxy, and in future works similar proxies will be needed. Perhaps common design studies can be grouped by their characteristics and those design study groups can be the proxies for the metrics to be calculated. To perform such a research work, previous examples must be accessible and available for study, which can be difficult as many system of systems studies involving DoDAF have a military context.

A second idea is to take a similar approach to the cross-validation but apply it in a reverse fashion. Instead of finding maximal overlap in modeling elements and

architecture elements, the goal can be set to minimal overlap and create hybrid-simulations, where two types of models are working together in a single simulation. Hybrid-simulations enable studying phenomena at varying scales bringing detail to high-level models and speed to high-detail models. Alternatively, hybrid models can model discrete and continuous phenomena in a consistent simulation. Either way, the lack of overlap of architecture elements used for one type of model compared to another type of model could be useful in identifying the opportunities for hybrid modeling. Difficulties may arise from the fact that in many cases the architecture views do not distinguish such discrete versus continuous phenomena in explicit ways. Therefore an architecture view can be useful for both continuous or discrete time modeling. This idea require refinement and further investigation.

Another idea is the expansion of DoDAF with views that are not standard. DoDAF already has a mechanism to include non-standard views under the category: fit for purpose. These fit for purpose views were ignored for the selection of modeling types because one program's fit for purpose views are different from another program's and therefore they cannot be used in a general method for selecting models. However, some fit for purpose views can be standardized by preserving generality.

For example, most systems of systems operate in a geographically diverse regions and can have pieces spread over large areas. A geographical map that shows the distribution of systems can help modeling and determining context greatly. Some architectures include OV-1s that serve this purpose; however, maps have many opportunities to be standardized and annotated with legends, projections, and colors. Using geographic information systems frameworks interactive and digital maps can be generated for systems of systems with various system and operational area informations layered on top of the maps. Dynamic simulations can be saved on top of such maps so that decision makers can "play back scenarios" and observe the effectiveness of designs.

A related thought about the usage of architecture views not as simple inputs to modeling selection and building but also as outputs of the models or as the visualizations of the outputs. One such example was developed and given in Section 8.3.1.2 in Figures 133 and 134. There is an exciting opportunity with this approach: the creation of more detailed views such as an OV-6c or SV-10c from less detailed views through modeling. The architects can focus on the higher level definitions and include only such low detail views in their work, and modelers can take that information and create higher-detail scenarios, event histories, etc. for architects to validate. In a way, computer modeling becomes a part of the architecting work and both phases are performed in a more tightly coupled iterative manner. This will most likely increase the quality of both architecture as well as the modeling products.

Once models are created, the architects can look at their outputs in a familiar format and validate computer models by inspecting them in a convenient way. These inspections can be performed within the modeling team as well for debugging purposes. Finally, the architects can keep some of the high-detail views from the modeling team and use them for a test for the modeling quality. Because the modeling team did not have access to the views, the similitude of their models to the architected system of systems can be tested fairly. Once the models are verified and validated, they can be used to create more architecture views (observations) and decision makers can confidently base their decisions on their outcomes.

Each of these future research ideas will be pursued as opportunities present themselves. Some were briefly introduced and examples generated but none were developed to the level of detail necessary in this work because they did not align perfectly well with the goals of the study.

9.7 Final conclusions

In conclusion, architectures are shown to be one step away from being turned into conceptual models and this step is simplified via the element maps. It is also shown that no single model can model a practical system of systems alone. To support these claims the element maps were created. While the research arguments gave form and rigor to the work, this work's main contribution is the creation of the element maps and discussion of their uses. Using the element maps, engineers can simplify their architecture to model translation efforts.

The main goal of the work as discussed in Section 3.2 was to help the modeler make decisions on the modeling type to pursue based on existing architecture views. This goal was achieved via the recommended workflow given in Section 9.5. Once a modeling type is selected, the modeler can then also follow the process shown in Figure 141 in Section 9.1 to construct a model. In that process, element maps play a large role. Appendix A includes these element maps that were constructed and tested. Unfortunately, specific recommendations cannot be given to modelers based on each possible architecture view combinations and each possible modeling type. Therefore, a general flow of work steps are given along with a few examples that were discussed in Chapter 8 to serve as guidance.

APPENDIX A

ARCHITECTURE ELEMENTS TO MODEL ELEMENTS MAPS

The tables here are compiled from the tables developed in Chapters 6 and 7 to be used in further analyses. It is significantly more convenient to have the tables compiled together instead of referencing many smaller tables. Using these tables, the reader can decide which modeling types to use for a given set a architecture views or decide on which architecture view to develop for specific types of modeling.

Table 204: Element maps for Graph, Probability, and System Dynamics models

Arch. element		Graph		Prob.	System dynamics		
View	Element	Vertex	Edge	Cond.	Stock	Flow	Var.
OV-1	System	Y	N	N	Y	N	M
	Action	N	Y	Y	N	Y	N
	Facility	Y	N	N	Y	N	M
OV-2	Op. node	Y	N	M	M	N	N
	Needline	N	Y	M	N	M	N
OV-3	Activity	Y	N	M	M	N	N
	Info	N	Y	M	N	M	N
OV-4	Org.	Y	N	N	M	N	N
	Relation	N	Y	N	N	M	M
OV-5a	Activity	M	N	N	N	N	N
	Relation	N	M	N	N	N	N
OV-5b	Activity	Y	N	Y	Y	N	N
	I/O	N	Y	Y	N	Y	N
	Misc.	N	N	M	N	N	Y
OV-6a	Activity	N	N	N	N	N	N
	Relation	N	N	N	N	N	N
	Rules	N	M	N	N	N	N
OV-6b	State	Y	N	Y	Y	N	N
	Activity	N	N	M	N	N	Y
	Trans.	N	Y	M	N	Y	N
OV-6c	Activity	Y	N	N	N	N	N
	Timeline	N	N	N	N	N	N
	Event	N	Y	Y	N	N	N
SV-1	System	N	N	N	Y	N	N
	Node	Y	N	N	Y	N	N
	Int.face	N	Y	M	N	Y	N
SV-2	System	Y	N	Y	Y	N	N
	Port	N	Y	N	N	N	Y
	Flowline	Y	N	Y	N	Y	N
SV-3	System	Y	N	Y	Y	N	N
	Resource	N	Y	Y	N	Y	M
SV-4	Function	Y	N	Y	Y	N	N
	Func.I/O	N	Y	N	N	Y	N
	Store	M	N	N	Y	N	N
SV-5a	Function	Y	N	N	N	N	N
	Activity	Y	N	N	N	N	N
SV-5b	System	Y	N	N	N	N	N
	Activity	Y	N	N	N	N	N
SV-6	System	Y	N	Y	Y	N	M
	Resource	N	Y	Y	N	Y	Y
SV-7	Metric	N	M	M	N	N	M
SV-8	M.stone	N	N	N	N	N	N
	Time	N	N	N	N	N	N
SV-9	System	Y	N	N	N	N	N
	Forecast	Y	N	N	N	N	N
SV-10a	Logic	N	N	N	N	N	N
SV-10b	State	Y	N	N	M	N	N
	Function	N	Y	Y	N	M	M
SV-10c	System	M	N	N	N	N	N
	Function	M	N	N	N	N	N
	Interact	N	M	N	N	M	N

Table 205: Element maps for Petri Net and Queueing models

Arch. element		Petri net			Queueing		
View	Element	Place	Trans.	Arc	Arrival	Size	Server
OV-1	System	Y	N	N	M	M	Y
	Action	N	Y	Y	Y	M	N
	Facility	Y	N	N	N	N	Y
OV-2	Op. node	M	N	N	N	N	N
	Needline	N	M	M	N	N	N
OV-3	Activity	M	N	N	N	N	M
	Info	N	M	M	M	M	N
OV-4	Org.	N	N	N	N	N	N
	Relation	N	N	N	N	N	N
OV-5a	Activity	N	N	N	N	N	N
	Relation	N	N	N	N	N	N
OV-5b	Activity	Y	N	N	N	Y	N
	I/O	N	Y	Y	Y	N	N
	Misc.	N	N	N	N	N	M
OV-6a	Activity	M	M	N	N	N	N
	Relation	N	N	M	N	N	N
	Rules	Y	Y	Y	N	N	N
OV-6b	State	Y	N	N	N	N	N
	Activity	N	Y	N	N	N	Y
	Trans.	N	N	Y	Y	N	N
OV-6c	Activity	Y	N	N	N	Y	Y
	Timeline	N	N	N	N	M	M
	Event	N	Y	Y	Y	M	M
SV-1	System	M	N	N	N	N	Y
	Node	M	N	N	N	N	Y
	Int.face	N	N	M	M	M	N
SV-2	System	Y	N	N	N	N	Y
	Port	N	N	Y	M	M	Y
	Flowline	N	M	N	Y	Y	N
SV-3	System	Y	N	N	N	N	Y
	Resource	N	Y	Y	M	M	N
SV-4	Function	N	Y	N	N	N	Y
	Func.I/O	Y	N	Y	M	Y	N
	Store	Y	N	N	Y	M	N
SV-5a	Function	N	N	N	N	N	N
	Activity	N	N	N	N	N	N
SV-5b	System	N	N	N	N	N	M
	Activity	N	N	N	N	N	N
SV-6	System	Y	N	N	N	N	Y
	Resource	N	Y	Y	Y	Y	N
SV-7	Metric	N	M	N	M	M	N
SV-8	Mi.stone	N	N	N	N	N	N
	Time	N	N	N	N	N	N
SV-9	System	N	N	N	N	N	N
	Forecast	N	N	N	N	N	N
SV-10a	Logic	Y	Y	Y	N	N	N
SV-10b	State	Y	N	N	N	N	N
	Function	N	Y	Y	N	N	N
SV-10c	System	N	N	N	N	N	Y
	Function	Y	N	N	N	Y	N
	Interact	N	Y	Y	Y	N	N

Table 206: Element maps for Markov Chain and Agent-based models

Arch. element		Markov chain		Agent-based			
View	Element	State	Trans.	Agent	Enviro.	Int.act.	Rule
OV-1	System	N	N	M	N	N	N
	Action	N	Y	N	N	M	N
	Facility	Y	N	M	Y	N	N
OV-2	Op. node	M	N	M	N	N	M
	Needline	N	M	N	N	N	M
OV-3	Activity	M	N	M	N	M	M
	Info	N	M	N	N	M	M
OV-4	Org.	M	M	Y	N	N	N
	Relation	M	M	N	N	Y	Y
OV-5a	Activity	N	N	N	N	N	M
	Relation	N	N	N	N	N	N
OV-5b	Activity	Y	N	N	N	N	Y
	I/O	N	Y	N	N	N	N
	Misc.	N	N	M	N	N	N
OV-6a	Activity	N	N	N	N	N	N
	Relation	N	N	N	N	N	N
	Rules	N	N	N	Y	N	Y
OV-6b	State	Y	N	N	M	N	M
	Activity	N	N	N	N	N	Y
	Trans.	N	Y	N	N	Y	Y
OV-6c	Activity	M	N	N	N	N	M
	Timeline	N	N	N	N	N	N
	Event	N	M	N	N	Y	Y
SV-1	System	N	N	Y	N	N	N
	Node	N	N	M	M	N	M
	Int.face	N	M	N	N	Y	M
SV-2	System	Y	N	Y	N	N	N
	Port	N	N	Y	N	Y	N
	Flowline	N	Y	N	N	Y	M
SV-3	System	Y	N	Y	N	N	N
	Resource	N	Y	N	N	Y	M
SV-4	Function	M	N	N	N	N	Y
	Func.I/O	N	M	N	N	Y	N
	Store	M	N	M	M	N	N
SV-5a	Function	N	N	N	N	N	M
	Activity	N	N	N	N	N	M
SV-5b	System	N	N	N	N	N	M
	Activity	N	N	N	N	N	M
SV-6	System	Y	N	Y	N	N	N
	Resource	N	Y	N	N	Y	N
SV-7	Metric	N	M	N	N	M	M
SV-8	M.stone	N	N	N	N	N	N
	Time	N	N	N	N	N	N
SV-9	System	N	N	M	N	N	N
	Forecast	N	N	N	N	M	M
SV-10a	Logic	N	N	M	M	Y	Y
SV-10b	State	M	N	Y	N	M	Y
	Function	N	M	Y	N	Y	Y
SV-10c	System	Y	N	Y	M	N	N
	Function	Y	N	N	N	N	Y
	Interact	N	Y	N	N	Y	Y

Table 207: Element maps for Discrete Event models

Arch. element		Discrete Event					
View	Element	Event	Queue	Trans.	Server	Entity	Res.
OV-1	System	N	N	N	M	Y	N
	Action	Y	M	N	N	N	N
	Facility	N	N	N	M	N	M
OV-2	Op. node	M	M	N	M	N	N
	Needline	N	N	M	N	M	N
OV-3	Activity	M	M	N	M	N	N
	Info	M	M	M	N	M	M
OV-4	Org.	N	N	N	N	N	N
	Relation	N	N	N	N	N	N
OV-5a	Activity	N	N	N	N	N	N
	Relation	N	N	N	N	N	N
OV-5b	Activity	Y	N	N	N	N	N
	I/O	N	Y	Y	N	M	N
	Misc.	N	N	N	M	N	M
OV-6a	Activity	N	N	N	N	N	N
	Relation	N	N	N	N	N	N
	Rules	N	Y	N	M	M	N
OV-6b	State	N	Y	N	N	M	N
	Activity	N	N	N	Y	N	N
	Trans.	Y	N	Y	N	N	N
OV-6c	Activity	N	N	N	Y	N	N
	Timeline	N	M	N	N	N	N
	Event	Y	M	Y	N	N	M
SV-1	System	M	M	N	M	M	N
	Node	M	M	N	M	M	N
	Int.face	N	N	Y	N	M	Y
SV-2	System	N	N	N	Y	M	N
	Port	M	M	N	N	N	N
	Flowline	N	N	Y	N	M	Y
SV-3	System	M	N	N	Y	Y	N
	Resource	N	N	Y	N	M	Y
SV-4	Function	M	N	N	Y	M	N
	Func.I/O	N	M	M	N	Y	Y
	Store	M	N	N	Y	Y	Y
SV-5a	Function	N	N	N	N	N	N
	Activity	N	N	N	N	N	N
SV-5b	System	N	N	N	M	N	N
	Activity	N	N	N	N	N	N
SV-6	System	Y	N	N	Y	M	N
	Resource	Y	Y	Y	N	Y	Y
SV-7	Metric	M	M	N	M	M	M
SV-8	M.stone	N	N	N	N	N	N
	Time	N	N	N	N	N	N
SV-9	System	N	N	N	N	N	N
	Forecast	N	N	N	N	N	N
SV-10a	Logic	N	M	N	N	N	N
SV-10b	State	N	M	N	N	Y	N
	Function	Y	N	Y	M	N	M
SV-10c	System	N	N	N	Y	Y	N
	Function	Y	M	N	N	N	M
	Interact	N	N	Y	N	M	M

APPENDIX B

ALGORITHMS USED IN THE EXPERIMENTS

B.1 Discrete event simulations

```
from random import betavariate, random, seed
from operator import itemgetter
from time import clock
from copy import deepcopy

executestarttime = clock()
def myintegratorandplotifier(pairs, divisor):
    val = 0
    for i in range(len(pairs)):
        time = pairs[i][0]
        if i == 0 :
            integral = [(time, val)]
            val += pairs[i][1]
            integral.append((time, val))
        else:
            if time == integral[-1][0]:
                val += pairs[i][1]
                integral[-1] = (time, val)
            else:
                integral.append((time, val))
                val += pairs[i][1]
                integral.append((time, val))
    l = reversed(range(len(integral)-2))
    for i in l:
        checkone = integral[i+1][1] == integral[i+2][1]
        checktwo = integral[i+1][1] == integral[i][1]
        if checkone and checktwo: integral.pop(i+1)
    result = [ (t,float(v)/divisor) for (t,v) in integral ]
    return result
def logger(**args):
    objlist = []
    #time, entid, enttype, funcname, servid, servtype, success, comment
```

```
33      entrylist = [world.clock,-1,'none','',-1,'none',False,'']
34      if 'entity' in args:
35          objlist.append(args['entity'])
36          entrylist[1]=args['entity'].id
37          entrylist[2]=args['entity'].type
38      if 'function' in args:
39          objlist.append(args['function'])
40          entrylist[3]=args['function'].name
41      if 'server' in args:
42          objlist.append(args['server'])
43          entrylist[4]=args['server'].id
44          entrylist[5]=args['server'].type
45      entrylist[6]=args['success']
46      if 'comment' in args: entrylist[7]=args['comment']
47      if 'special' in args: entrylist[3]=entrylist[3]+args['special']
48      for obj in objlist: obj.log.append(tuple(entrylist))
49  def historian(obj, **args):
50      if 'queue' in args: obj.queue.append((args['time'], args['queue']))
51      if 'active' in args: obj.active.append((args['time'], args['active']))
52  class Entity():
53      def __init__(self, entitytype, priority=0):
54          self.id = len(world.entitycensus)
55          if self not in world.entitycensus: world.entitycensus.append(self)
56          self.type = entitytype
57          self.priority = priority
58          self.alive = True
59          self.flag = {}
60          for process in world.processcensus: self.flag[process.name] = 0
61          self.log = []
62          logger(entity=self, success=True, special='arrival')
63  class Server():
64      def __init__(self, servertype):
65          self.id = len(world.servercensus)
66          if self not in world.servercensus: world.servercensus.append(self)
67          self.type = servertype
68          self.busyuntil = 0
69          self.active = []
70          self.log = []
71  class Event():
72      def __init__(self, name, serverdict):
73          self.name=name
```

```
74          if self not in world.processcensus: world.processcensus.append(self)
75          self.serverdict=serverdict
76          self.queue = []
77          self.active = []
78          self.log=[]
79          self.numarr = 0
80      def process(self, entity, server):
81          [a,b,l,u]=self.serverdict[server.type][1]
82          duration = l + (u-l) * betavariate(a,b)
83          server.busyuntil = world.clock + duration
84          historian(self, queue=-1, active=+1, time=world.clock)
85          historian(server, active=+1, time=world.clock)
86          success = random() < self.serverdict[server.type][0]
87          logger(entity=entity, function=self, server=server, success=success)
88          if success:
89              for elem in world.processflow[self.name]:
90                  world.schedule(time=world.clock+duration,function=world.enqueue,
91                                  entity=entity, enqfunction=elem)
92          else:
93              entity.flag[self.name] -= 1
94              world.schedule(time=world.clock+duration, function=world.enqueue,
95                                  entity=entity, enqfunction=self)
96          historian(self, active=-1, time=world.clock+duration)
97          historian(server, active=-1, time=world.clock+duration)
98          world.schedule(time=world.clock+duration, function=world.decide)
99  class EventC(Event):
100     def process(self, entity, server):
101         [a,b,l,u]=self.serverdict[server.type][1]
102         duration = l + (u-l) * betavariate(a,b)
103         server.busyuntil = world.clock + duration
104         historian(self, queue=-1, active=+1, time=world.clock)
105         historian(server, active=+1, time=world.clock)
106         success = random() < self.serverdict[server.type][0]
107         logger(entity=entity, function=self, server=server, success=success)
108         if success:
109             if random() < 0.5:
110                 world.schedule(time=world.clock+duration,
111                                 function=world.enqueue,
112                                 entity=entity,
113                                 enqfunction=world.processflow[self.name][0])
114             else:
```

```
115                 world.schedule(time=world.clock+duration,
116                                function=world.enqueue,
117                                entity=entity,
118                                enqfunction=world.processflow[self.name][1])
119         else:
120             entity.flag[self.name] -= 1
121             world.schedule(time=world.clock+duration, function=world.enqueue,
122                                entity=entity, enqfunction=self)
123         historian(self, active=-1, time=world.clock+duration)
124         historian(server, active=-1, time=world.clock+duration)
125         world.schedule(time=world.clock+duration, function=world.decide)
126 class Engage(Event):
127     def process(self, entity, server):
128         [a,b,l,u]=self.serverdict[server.type][1]
129         duration = l + (u-l) * betavariate(a,b)
130         server.busyuntil = world.clock + duration
131         historian(self, queue=-1, active=+1, time=world.clock)
132         historian(server, active=+1, time=world.clock)
133         success = random() < self.serverdict[server.type][0]
134         logger(entity=entity, function=self, server=server, success=success)
135         if success: entity.alive = False
136         for elem in world.processflow[self.name]:
137             world.schedule(time=world.clock+duration, function=world.enqueue,
138                            entity=entity, enqfunction=elem)
139         historian(self, active=-1, time=world.clock+duration)
140         historian(server, active=-1, time=world.clock+duration)
141         world.schedule(time=world.clock+duration, function=world.decide)
142 class Assess(Event):
143     def process(self, entity, server):
144         [a,b,l,u]=self.serverdict[server.type][1]
145         duration = l + (u-l) * betavariate(a,b)
146         server.busyuntil = world.clock + duration
147         historian(self, queue=-1, active=+1, time=world.clock)
148         historian(server, active=+1, time=world.clock)
149         success = random() < self.serverdict[server.type][0]
150         logger(entity=entity, function=self, server=server, success=success)
151         if success:
152             if entity.alive: entrypointprocessname = 'arrival'
153             else: entrypointprocessname = self.name
154             for elem in world.processflow[entrypointprocessname]:
155                 world.schedule(time=world.clock+duration, entity=entity,
```

```
156                                 function=world.enqueue, enqfunction=elem)
157         else:
158             entity.flag[self.name] -= 1
159             world.schedule(time=world.clock+duration, function=world.enqueue,
160                                 entity=entity, enqfunction=self)
161         historian(self, active=-1, time=world.clock+duration)
162         historian(server, active=-1, time=world.clock+duration)
163         world.schedule(time=world.clock+duration, function=world.decide)
164 class Simulation():
165     def __init__(self, history=[]):
166         self.fel = []
167         self.clock = 0
168         self.queue = []
169         self.processflow={}
170         self.processcensus=[]
171         self.servercensus=[]
172         self.entitycensus=[]
173         self.departedlist=[]
174         self.history=history
175         self.result={}
176     def advanceclock(self):
177         minval = min(self.fel, key=itemgetter(0))
178         mins = [(i, v) for i, v in enumerate(self.fel) if v[0] == minval[0]]
179         for elem in mins:
180             if elem[1][1]['function'] == world.enqueue:
181                 world.clock, output = self.fel.pop(elem[0])
182                 return output
183         world.clock, output = self.fel.pop(mins[0][0])
184         return output
185     def decide(self, **possibleuselessdictionary):
186         i = 0
187         while i < len(self.queue):
188             if self.queue[i][1].log[-1][0] < self.clock - 1000:
189                 priority, entity, function = self.queue.pop(i)
190                 for process in self.processcensus: entity.flag[process.name] = 0
191                 for arrprocess in self.processflow['arrival']:
192                     self.enqueue(entity=entity, enqfunction=function)
193             else:
194                 i += 1
195         waiting=self.queue[:] # this is a shallow-copy!
196         idles = [i for i in self.servercensus if i.busyuntil <= self.clock]
```

```
197         idlesnames = [i.type for i in idles]
198         while idles != [] and waiting != []:
199             possibleserverdict = waiting[-1][2].serverdict.copy()
200             match = False
201             idlenum = ''
202             OEC={}
203             for elem in possibleserverdict:
204                     PoS=possibleserverdict[elem][0]
205                     a,b,l,u=possibleserverdict[elem][1]
206                     meantime=l+(a/(a+b))*(u-l)
207                     OEC[elem] = 0.5*PoS + 0.5/meantime
208             while not match and possibleserverdict!={}:
209                 k, v = max(OEC.items(), key=lambda x:x[1])
210                 if k in idlesnames:
211                     idlenum = idlesnames.index(k)
212                     idlesnames.pop(idlenum)
213                     server=idles.pop(idlenum)
214                     nomatch = False
215                 else:
216                     del possibleserverdict[k]
217                     del OEC[k]
218             if idlenum != '':
219                 [x,entity,event]=self.queue.pop(self.queue.index(waiting[-1]))
220                 event.process(entity=entity, server=server)
221             waiting.pop()
222     def enqueue(self, entity, enqfunction, **uselessotherinputs):
223         if enqfunction == 'depart':
224             self.departedlist.append(entity)
225             return
226         entity.flag[enqfunction.name] = entity.flag[enqfunction.name]+1
227         self.queue.append((entity.priority,entity,enqfunction))
228         self.queue.sort(key=itemgetter(0), reverse=True)
229         self.schedule(function=self.decide)
230         logger(entity=entity, function=enqfunction, success=True,
231                 special='Queue', comment='entered queue')
232         historian(enqfunction, time=self.clock, queue=+1)
233     def schedule(self, **args):
234         if 'time' in args: time=args['time']
235         else: time=self.clock
236         self.fel.append((time, args))
237     def simulate(self, repetitions, altname):
```

```
238         rep=1
239         while rep <= repetitions:
240             repstarttime = clock()
241             self.start(altname)
242             seed(rep-1)
243             while self.fel != [] and self.clock < 50:
244                 try:
245                     print(self.servercensus[0].busyuntil > self.clock)
246                     print(self.processcensus[0].queue[-1])
247                     print(self.servercensus[1].busyuntil > self.clock)
248                     print(self.processcensus[1].queue[-1])
249                     print(self.servercensus[2].busyuntil > self.clock)
250                     print(self.processcensus[2].queue[-1])
251                 except:
252                     pass
253                 print(self.clock)
254                 command = self.advanceclock()
255                 command.pop('function')(**command)
256             self.caselog(self.clock)
257             self.__init__(history=self.history)
258             rep += 1
259             print('This run finished in '+str(clock()-repstarttime)+' seconds.')
260         self.timehistory()
261         self.output(altname)
262     def output(self, altname):
263         times = [ i[0] for i in self.history ]
264         mintime = min(times)
265         avgtime = float(sum(times)) / len(times)
266         maxtime = max(times)
267         vartime = (sum([(t-avgtime)**2 for t in times])/(len(times)-1))
268         sdtime  = vartime**0.5
269         setime  = (vartime/len(times))**0.5
270         lcl     = ''
271         ucl     = ''
272         stats = open(altname+'_stats.csv', 'w')
273         stats.write(','.join(['Number of repetitions','Minimum finish time',
274                               'Average finish time','Maximum finish time',
275                               'Finish time variance',
276                               'Finish time standard deviation',
277                               'Finish time standard error',
278                               'UCL of finish time',
```

```
279                                 'LCL of finish time'])+'\n')
280             stats.write(','.join([str(len(times)),str(mintime),str(avgtime),
281                                 str(maxtime),str(vartime),str(sdtime),str(setime),
282                                 str(lcl),str(ucl)])+'\n')
283             stats.write('\n')
284             stats.write('Time, Probability to finish\n')
285             stats.write('0,0\n')
286             stats.write('\n'.join([','.join([str(sorted(times)[i]),
287                                         str(float(i+1)/len(self.history))])
288                                  for i in range(len(times))]))
289             stats.close()
290             hist = open(altname+'_history.csv', 'w')
291             m=max([len(i) for i in self.result['server active history'].values()])
292             lines = ['' for i in range(m+1)]
293             lines[0] = ','.join([i+'_time,'+i+'_busy' for i in
294                                 self.result['server active history'].keys()])
295             hist.write(lines[0]+'\n')
296             for l in range(m):
297                 for v in self.result['server active history'].values():
298                     try:
299                         if lines[l+1] == '':
300                             lines[l+1]=str(v[l][0])+','+str(v[l][1])
301                         else:
302                             lines[l+1]=lines[l+1]+','+str(v[l][0])+','+str(v[l][1])
303                     except:
304                         if lines[l+1] == '':
305                             lines[l+1] = ','
306                         else:
307                             lines[l+1] = lines[l+1]+',,'
308                 hist.write(lines[l+1]+'\n')
309             hist.close()
310         def caselog(self, fintime):
311             logstarttime=clock()
312             servhist = []
313             for server in self.servercensus:
314                 server.active.sort(key=itemgetter(0))
315                 servhist.append((server.id,server.type,deepcopy(server.active)))
316             evnthist = []
317             for event in self.processcensus:
318                 event.queue.sort(key=itemgetter(0))
319                 event.active.sort(key=itemgetter(0))
```

```
320             evnthist.append((event.name,deepcopy(event.queue),
321                              deepcopy(event.active)))
322         self.history.append((fintime,servhist,evnthist))
323         print('Logging took '+str(clock()-logstarttime)+' seconds.')
324     def postprocess(self):
325         ''' This function makes sure the simulation attains desired
326             level of accuracy. '''
327         pass
328     def timehistory(self):
329         timehistorystarttime = clock()
330         servacthist, evntquehist, evntacthist, count = {}, {}, {}, {}
331         for run in self.history:
332             for elem in run[1]:
333                 if elem[1] not in servacthist.keys(): servacthist[elem[1]] = []
334                 if elem[1] not in count.keys(): count[elem[1]] = 0
335                 count[elem[1]] += 1
336                 servacthist[elem[1]].extend(deepcopy(elem[2]))
337             for elem in run[2]:
338                 if elem[0] not in evntquehist.keys(): evntquehist[elem[0]] = []
339                 if elem[0] not in evntacthist.keys(): evntacthist[elem[0]] = []
340                 if elem[0] not in count.keys(): count[elem[0]] = 0
341                 count[elem[0]] += 1
342                 evntquehist[elem[0]].extend(deepcopy(elem[1]))
343                 evntacthist[elem[0]].extend(deepcopy(elem[2]))
344         for key, value in servacthist.items():
345             value.sort(key=itemgetter(0))
346             servacthist[key]=myintegratorandplotifier(value,count[key])
347         for key, value in evntquehist.items():
348             value.sort(key=itemgetter(0))
349             evntquehist[key]=myintegratorandplotifier(value,len(self.history))
350         for key, value in evntacthist.items():
351             value.sort(key=itemgetter(0))
352             evntacthist[key]=myintegratorandplotifier(value,count[key])
353         self.result['server active history'] = servacthist
354         self.result['process queue history'] = evntquehist
355         self.result['process active history'] = evntacthist
356         print(evntacthist)
357         self.result['counts'] = count
358         print('History took '+str(clock()-timehistorystarttime)+' seconds.')
359     def start(self, altname):
360         exec(open(altname+'.txt').read())
```

```
361         for n in range(50):exec('Target_'+str(n)+'=Entity("Target_'+str(n)+'")')
362         for proc in self.processcensus:
363             proc.numarr=len([1 for i in self.processflow.values() if proc in i])
364         for entity in self.entitycensus:
365             for arrprocess in self.processflow['arrival']:
366                 self.enqueue(entity=entity, enqfunction=arrprocess)
367 ''' Test Code '''
368 world = Simulation()
369 world.simulate(2, 'ComparisonRunTrial')
370 print('Total execution time was '+str(clock()-executestarttime)+' seconds.')
```

Listing B.1: Discrete event simulation using the modified HADES code in Python

B.2 Agent-based simulations

```
1  """ This is an attempt at building an agent-based model for the RWDC
2      problem using Dr. Jones Wyatt's DoDAF viewpoints."""
3  # import random
4  # from copy import copy
5  from mesa import Agent, Model
6  from mesa.time import RandomActivation
7  #
8  class System(Agent):
9      """Systems that communicate with each other and change data are
10         modeled as agents."""
11     def __init__(self, name, model, gen_data=None):
12         super().__init__(model.create_agent(self), model)
13         self.model = model
14         self.name = name # these must be unique!
15         self.model.add_system(self)
16         self.interfaces = {}
17         self.data_generated = gen_data
18         self.data_manipulations = {}
19         self.incoming_data = set()
20         self.outgoing_data = set()
21     def add_interface_to(self, interface, data_desc):
22         if data_desc in self.interfaces:
23             self.interfaces[data_desc].append(interface)
24         else:
25             self.interfaces[data_desc] = [interface]
26     def add_data_manipulation(self, input_data_desc, output_data_desc):
27         if input_data_desc in self.data_manipulations:
```

```
28              self.data_manipulations[input_data_desc].add(output_data_desc)
29          else:
30              self.data_manipulations[input_data_desc] = {output_data_desc} # this is a
        set
31      def send_data(self, data_desc):
32          for interface in self.interfaces[data_desc]:
33              interface.load_data(data_desc)
34      def receive_data(self, data_desc): # do something with the source?
35          self.incoming_data.add(data_desc)
36      def act_on_data(self, data_desc):
37          for new_data_desc in self.data_manipulations[data_desc]:
38              self.outgoing_data.add(new_data_desc)
39      def generate_data(self):
40          if self.data_generated:
41              self.outgoing_data.add(self.data_generated)
42      def step(self):
43          for data_desc in self.incoming_data:
44              self.act_on_data(data_desc)
45          self.generate_data()
46          for data_desc in self.outgoing_data:
47              self.send_data(data_desc)
48          self.incoming_data = set()
49          self.outgoing_data = set()
50      #
51  #
52  class Interface(Agent):
53      """Data communications between agents"""
54      def __init__(self, data_desc, source, sink, model):
55          super().__init__(model.create_agent(self), model)
56          self.model = model
57          self.data_desc = data_desc
58          self.source = model.systems[source]
59          self.sink = model.systems[sink]
60          self.loaded_data = None
61          self.source.add_interface_to(self, self.data_desc)
62      def load_data(self, data):
63          self.loaded_data = data
64      def carry_data(self):
65          if self.loaded_data:
66              self.sink.receive_data(self.loaded_data)
67              self.loaded_data = None
```

```
68      def step(self):
69          self.carry_data()
70      #
71  #
72  class RWDCModel(Model):
73      """Execution logic for the model"""
74      def __init__(self):
75          super().__init__()
76          self.schedule = RandomActivation(self)
77          self.systems = {}
78          self.interfaces = []
79          self.data_types = []
80          self.unique_id_count = -1 # the first agent will get id=0
81          self.running = False
82      def create_agent(self, agent):
83          self.schedule.add(agent)
84          self.unique_id_count += 1
85          return self.unique_id_count
86      def add_system(self, system):
87          if system not in self.systems:
88              self.systems[system.name] = system
89      def add_interface(self, interface):
90          if interface not in self.interfaces:
91              self.interfaces.append(interface)
92      def execute(self, steps=5):
93          self.running = True
94          for stepno in range(steps):
95              print("model step " + str(stepno) + ".")
96              self.schedule.step()
97          #
98      #
99  #
100 # 1) Create the model first
101 MODEL = RWDCModel()
102 # 2) Create the systems
103 System("Pilot Workstation Computer", MODEL,
104     "Waypoints") # data gen not included in Jones Wyatt's views
105 System("Command Datalink Ground Transceiver", MODEL)
106 # System("Safety Pilot Flight Box", MODEL) Not in Jones Wyatt's SV-6
107 System("Video Datalink Ground Receiver", MODEL)
108 System("Sensor Payload Workstation Computer", MODEL)
```

```
System("Command Datalink UAV Transceiver", MODEL)
System("Sensor Payload", MODEL)
System("Video Datalink UAV Transmitter", MODEL, "Video File")
System("Flight Control System", MODEL, "UAV Position") # wasn't in SV-1
# 3) Create the interfaces
Interface("Waypoints", "Pilot Workstation Computer",
    "Command Datalink Ground Transceiver", MODEL)
Interface("UAV Position", "Command Datalink Ground Transceiver",
    "Pilot Workstation Computer", MODEL)
Interface("Waypoints", "Command Datalink Ground Transceiver",
    "Command Datalink UAV Transceiver", MODEL)
Interface("UAV Position", "Command Datalink UAV Transceiver",
    "Command Datalink Ground Transceiver", MODEL)
Interface("Pan/Tilt/Zoom", "Command Datalink Ground Transceiver",
    "Command Datalink UAV Transceiver", MODEL)
Interface("Sensor Orientation", "Command Datalink UAV Transceiver",
    "Command Datalink Ground Transceiver", MODEL)
Interface("Sensor Orientation", "Command Datalink Ground Transceiver",
    "Sensor Payload Workstation Computer", MODEL)
Interface("Pan/Tilt/Zoom", "Sensor Payload Workstation Computer",
    "Command Datalink Ground Transceiver", MODEL)
Interface("Waypoints", "Command Datalink UAV Transceiver",
    "Flight Control System", MODEL)
Interface("UAV Position", "Flight Control System",
    "Command Datalink UAV Transceiver", MODEL)
Interface("Pan/Tilt/Zoom", "Command Datalink UAV Transceiver",
    "Flight Control System", MODEL)
Interface("Sensor Orientation", "Flight Control System",
    "Command Datalink UAV Transceiver", MODEL)
Interface("Pan/Tilt/Zoom", "Flight Control System",
    "Sensor Payload", MODEL)
Interface("Sensor Orientation", "Sensor Payload",
    "Flight Control System", MODEL)
Interface("Video File", "Sensor Payload",
    "Video Datalink UAV Transmitter", MODEL)
Interface("Video File", "Video Datalink UAV Transmitter",
    "Video Datalink Ground Receiver", MODEL)
Interface("Video File", "Video Datalink Ground Receiver",
    "Sensor Payload Workstation Computer", MODEL)
# 4) Define data manipulations (this wasn in Jones Wyatt's views)
MODEL.systems["Pilot Workstation Computer"].add_data_manipulation(
```

```
150     "UAV Position", "Waypoints")
151 MODEL.systems["Command Datalink Ground Transceiver"].add_data_manipulation(
152     "Waypoints", "Waypoints")
153 MODEL.systems["Command Datalink Ground Transceiver"].add_data_manipulation(
154     "Waypoints", "Pan/Tilt/Zoom")
155 MODEL.systems["Command Datalink Ground Transceiver"].add_data_manipulation(
156     "Waypoints", "Sensor Orientation")
157 MODEL.systems["Command Datalink Ground Transceiver"].add_data_manipulation(
158     "Waypoints", "UAV Position")
159 MODEL.systems["Command Datalink Ground Transceiver"].add_data_manipulation(
160     "Pan/Tilt/Zoom", "Waypoints")
161 MODEL.systems["Command Datalink Ground Transceiver"].add_data_manipulation(
162     "Pan/Tilt/Zoom", "Pan/Tilt/Zoom")
163 MODEL.systems["Command Datalink Ground Transceiver"].add_data_manipulation(
164     "Pan/Tilt/Zoom", "Sensor Orientation")
165 MODEL.systems["Command Datalink Ground Transceiver"].add_data_manipulation(
166     "Pan/Tilt/Zoom", "UAV Position")
167 MODEL.systems["Command Datalink Ground Transceiver"].add_data_manipulation(
168     "Sensor Orientation", "Waypoints")
169 MODEL.systems["Command Datalink Ground Transceiver"].add_data_manipulation(
170     "Sensor Orientation", "Pan/Tilt/Zoom")
171 MODEL.systems["Command Datalink Ground Transceiver"].add_data_manipulation(
172     "Sensor Orientation", "Sensor Orientation")
173 MODEL.systems["Command Datalink Ground Transceiver"].add_data_manipulation(
174     "Sensor Orientation", "UAV Position")
175 MODEL.systems["Command Datalink Ground Transceiver"].add_data_manipulation(
176     "UAV Position", "Waypoints")
177 MODEL.systems["Command Datalink Ground Transceiver"].add_data_manipulation(
178     "UAV Position", "Pan/Tilt/Zoom")
179 MODEL.systems["Command Datalink Ground Transceiver"].add_data_manipulation(
180     "UAV Position", "Sensor Orientation")
181 MODEL.systems["Command Datalink Ground Transceiver"].add_data_manipulation(
182     "UAV Position", "UAV Position")
183 MODEL.systems["Sensor Payload Workstation Computer"].add_data_manipulation(
184     "Video File", "Pan/Tilt/Zoom")
185 MODEL.systems["Sensor Payload Workstation Computer"].add_data_manipulation(
186     "Sensor Orientation", "Pan/Tilt/Zoom")
187 MODEL.systems["Video Datalink Ground Receiver"].add_data_manipulation(
188     "Video File", "Video File")
189 MODEL.systems["Video Datalink UAV Transmitter"].add_data_manipulation(
190     "Video File", "Video File")
```

```
MODEL.systems["Sensor Payload"].add_data_manipulation(
  "Pan/Tilt/Zoom", "Video File")
MODEL.systems["Sensor Payload"].add_data_manipulation(
  "Pan/Tilt/Zoom", "Sensor Orientation")
MODEL.systems["Flight Control System"].add_data_manipulation(
  "Pan/Tilt/Zoom", "Pan/Tilt/Zoom")
MODEL.systems["Flight Control System"].add_data_manipulation(
  "Pan/Tilt/Zoom", "UAV Position")
MODEL.systems["Flight Control System"].add_data_manipulation(
  "Pan/Tilt/Zoom", "Sensor Orientation")
MODEL.systems["Flight Control System"].add_data_manipulation(
  "Waypoints", "Pan/Tilt/Zoom")
MODEL.systems["Flight Control System"].add_data_manipulation(
  "Waypoints", "UAV Position")
MODEL.systems["Flight Control System"].add_data_manipulation(
  "Waypoints", "Sensor Orientation")
MODEL.systems["Flight Control System"].add_data_manipulation(
  "Sensor Orientation", "Pan/Tilt/Zoom")
MODEL.systems["Flight Control System"].add_data_manipulation(
  "Sensor Orientation", "UAV Position")
MODEL.systems["Flight Control System"].add_data_manipulation(
  "Sensor Orientation", "Sensor Orientation")
MODEL.systems["Command Datalink UAV Transceiver"].add_data_manipulation(
  "Sensor Orientation", "Sensor Orientation")
MODEL.systems["Command Datalink UAV Transceiver"].add_data_manipulation(
  "Sensor Orientation", "UAV Position")
MODEL.systems["Command Datalink UAV Transceiver"].add_data_manipulation(
  "Sensor Orientation", "Pan/Tilt/Zoom")
MODEL.systems["Command Datalink UAV Transceiver"].add_data_manipulation(
  "Sensor Orientation", "Waypoints")
MODEL.systems["Command Datalink UAV Transceiver"].add_data_manipulation(
  "UAV Position", "Sensor Orientation")
MODEL.systems["Command Datalink UAV Transceiver"].add_data_manipulation(
  "UAV Position", "UAV Position")
MODEL.systems["Command Datalink UAV Transceiver"].add_data_manipulation(
  "UAV Position", "Pan/Tilt/Zoom")
MODEL.systems["Command Datalink UAV Transceiver"].add_data_manipulation(
  "UAV Position", "Waypoints")
MODEL.systems["Command Datalink UAV Transceiver"].add_data_manipulation(
  "Pan/Tilt/Zoom", "Sensor Orientation")
MODEL.systems["Command Datalink UAV Transceiver"].add_data_manipulation(
```

```
232     "Pan/Tilt/Zoom", "UAV Position")
233 MODEL.systems["Command Datalink UAV Transceiver"].add_data_manipulation(
234     "Pan/Tilt/Zoom", "Pan/Tilt/Zoom")
235 MODEL.systems["Command Datalink UAV Transceiver"].add_data_manipulation(
236     "Pan/Tilt/Zoom", "Waypoints")
237 MODEL.systems["Command Datalink UAV Transceiver"].add_data_manipulation(
238     "Waypoints", "Sensor Orientation")
239 MODEL.systems["Command Datalink UAV Transceiver"].add_data_manipulation(
240     "Waypoints", "UAV Position")
241 MODEL.systems["Command Datalink UAV Transceiver"].add_data_manipulation(
242     "Waypoints", "Pan/Tilt/Zoom")
243 MODEL.systems["Command Datalink UAV Transceiver"].add_data_manipulation(
244     "Waypoints", "Waypoints")
245 # 5) Execute the model
246 MODEL.execute()
247 print("run completed")
```

Listing B.2: Agent-based simulation for the RWDC example using the Mesa Framework in Python

```
1  from mesa import Agent, Model
2  from mesa.time import RandomActivation
3  from copy import copy
4  import random
5
6  class ATCSCC(Agent):
7      """ATCSCC"""
8      def __init__(self, model, unique_id):
9          self.model = model
10         self.unique_id = unique_id
11         self.doing = dict()
12         self.request(self.coordinate_with_NWS) #initial action
13     def step(self):
14         iterateoverthis = copy(self.doing)
15         for action, time_remaining in iterateoverthis.items():
16             if time_remaining:
17                 action(duration=self.doing[action])
18                 self.doing[action] -= 1
19             else:
20                 action(duration=self.doing[action], next=True)
21                 self.doing.pop(action)
```

```
22      def request(self, function):
23          self.doing[function]=function.__defaults__[0]-1
24      def coordinate_with_NWS(self, duration=15, next=False):
25          """Event 1"""
26          print("ATCSCC - Event 1")
27          if next:
28              self.request(self.telcon_for_determining_workload)
29      def telcon_for_determining_workload(self, duration=15, next=False):
30          """Event 2"""
31          if duration == 14:
32              for agent in self.model.schedule.agents:
33                  if isinstance(agent,SystemUserCommunity):
34                      agent.request(agent.provide_user_input)
35          print("ATCSCC - Event 2")
36          if next:
37              self.request(self.develop_flow_control_plan)
38      def develop_flow_control_plan(self, duration=30, next=False):
39          """Event 3"""
40          print("ATCSCC - Event 3")
41          if next:
42              self.request(self.telcon_for_planning)
43      def telcon_for_planning(self, duration=15, next=False):
44          """Event 4"""
45          if duration == 14:
46              for agent in self.model.schedule.agents:
47                  if isinstance(agent, ARTCCTMU):
48                      agent.request(agent.telcon_for_planning)
49                  elif isinstance(agent, TRACONTMU):
50                      agent.request(agent.telcon_for_planning)
51                  elif isinstance(agent, TowerTMU):
52                      agent.request(agent.telcon_for_planning)
53          print("ATCSCC - Event 4")
54          if next:
55              self.request(self.monitor_traffic_flow)
56      def monitor_traffic_flow(self, duration=-1):
57          """Event 6"""
58          if duration == -2:
59              self.request(self.monitor_traffic_flow)
60              for agent in self.model.schedule.agents:
61                  if isinstance(agent,SystemUserCommunity):
62                      agent.request(agent.monitor_traffic_flow)
```

```
63                  elif isinstance(agent,ARTCCTMU):
64                      agent.request(agent.monitor_and_adjust_flow)
65                      agent.request(agent.monitor_traffic_flow)
66                  elif isinstance(agent,TRACONTMU):
67                      agent.request(agent.monitor_traffic_flow)
68                  elif isinstance(agent,TowerTMU):
69                      agent.request(agent.monitor_traffic_flow)
70          print("ATCSCC - Event 6 continuous")
71      def initiate_delay_program(self, duration=15, next=False):
72          """Event 12"""
73          print("ATCSCC - Event 12")
74          if next:
75              print("END OF SIMULATION")
76  class SystemUserCommunity(Agent):
77      """System User Community"""
78      def __init__(self, model, unique_id):
79          self.model = model
80          self.unique_id = unique_id
81          self.doing = dict()
82      def step(self):
83          iterateoverthis = copy(self.doing)
84          for action, time_remaining in iterateoverthis.items():
85              if time_remaining:
86                  action(duration=self.doing[action])
87                  self.doing[action] -= 1
88              else:
89                  action(duration=self.doing[action], next=True)
90                  self.doing.pop(action)
91      def request(self, function):
92          self.doing[function]=function.__defaults__[0]-1
93      def provide_user_input(self, duration=15, next=False):
94          """Event 2"""
95          print("System User Community - Event 2")
96      def monitor_traffic_flow(self, duration=-1):
97          """Event 6"""
98          print("System User Community - Event 6 continuous")
99  class ARTCCTMU(Agent):
100     """ARTCC TMU"""
101     def __init__(self, model, unique_id):
102         self.model = model
103         self.unique_id = unique_id
```

```
104         self.doing = dict()
105     def step(self):
106         iterateoverthis = copy(self.doing)
107         for action, time_remaining in iterateoverthis.items():
108             if time_remaining:
109                 action(duration=self.doing[action])
110                 self.doing[action] -= 1
111             else:
112                 action(duration=self.doing[action], next=True)
113                 self.doing.pop(action)
114     def request(self, function):
115         self.doing[function]=function.__defaults__[0]-1
116     def telcon_for_planning(self, duration=15, next=False):
117         """Event 4"""
118         print("ARTCC TMU - Event 4")
119         if next:
120             self.request(self.advise_facility_management)
121     def advise_facility_management(self, duration=10, next=False):
122         """Event 5"""
123         print("ARTCC TMU - Event 5")
124     def monitor_traffic_flow(self, duration=-1):
125         """Event 6"""
126         print("ARTCC TMU - Event 6 continuous")
127     def monitor_and_adjust_flow(self, duration=-1):
128         """Event 7"""
129         if duration == -2:
130             self.request(self.monitor_and_adjust_flow)
131             self.request(self.receive_inclement_weather_warning)
132             for agent in self.model.schedule.agents:
133                 if isinstance(agent,TRACONTMU):
134                     agent.request(agent.monitor_and_adjust_flow)
135                 elif isinstance(agent,TowerTMU):
136                     agent.request(agent.monitor_and_adjust_flow)
137                 elif isinstance(agent,ATCSupervisor):
138                     agent.request(agent.monitor_and_adjust_flow)
139         print("ARTCC TMU - Event 7 continuous")
140     def receive_inclement_weather_warning(self, duration=1, next=False):
141         """Event 8"""
142         print("ARTCC TMU - Event 8")
143         if next:
144             self.request(self.send_increase_spacing_message)
```

```
145     def send_increase_spacing_message(self, duration=1, next=False):
146         """Event 9"""
147         for agent in self.model.schedule.agents:
148             if isinstance(agent,ATCSupervisor):
149                 agent.request(agent.coordinate_with_sectors)
150             #
151         print("ARTCC TMU - Event 9")
152     def coordinate_ground_stop_and_departure_delay(self, duration=1, next=False):
153         """Event 11"""
154         for agent in self.model.schedule.agents:
155             if isinstance(agent,ATCSCC):
156                 agent.request(agent.initiate_delay_program)
157         print("ARTCC TMU - Event 11")
158 class TRACONTMU(Agent):
159     """TRACON TMU"""
160     def __init__(self, model, unique_id):
161         self.model = model
162         self.unique_id = unique_id
163         self.doing = dict()
164     def step(self):
165         iterateoverthis = copy(self.doing)
166         for action, time_remaining in iterateoverthis.items():
167             if time_remaining:
168                 action(duration=self.doing[action])
169                 self.doing[action] -= 1
170             else:
171                 action(duration=self.doing[action], next=True)
172                 self.doing.pop(action)
173     def request(self, function):
174         self.doing[function]=function.__defaults__[0]-1
175     def telcon_for_planning(self, duration=15, next=False):
176         """Event 4"""
177         print("TRACON TMU - Event 4")
178         if next:
179             self.request(self.advise_facility_management)
180     def advise_facility_management(self, duration=10, next=False):
181         """Event 5"""
182         print("TRACON TMU - Event 5")
183     def monitor_traffic_flow(self, duration=-1):
184         """Event 6"""
185         print("TRACON TMU - Event 6 continuous")
```

```
186     def monitor_and_adjust_flow(self, duration=-1):
187         """Event 7"""
188         print("TRACON TMU - Event 7 continuous")
189 class TowerTMU(Agent):
190     """Tower TMU"""
191     def __init__(self, model, unique_id):
192         self.model = model
193         self.unique_id = unique_id
194         self.doing = dict()
195     def step(self):
196         iterateoverthis = copy(self.doing)
197         for action, time_remaining in iterateoverthis.items():
198             if time_remaining:
199                 action(duration=self.doing[action])
200                 self.doing[action] -= 1
201             else:
202                 action(duration=self.doing[action], next=True)
203                 self.doing.pop(action)
204     def request(self, function):
205         self.doing[function]=function.__defaults__[0]-1
206     def telcon_for_planning(self, duration=15, next=False):
207         """Event 4"""
208         print("Tower TMU - Event 4")
209         if next:
210             self.request(self.advise_facility_management)
211     def advise_facility_management(self, duration=10, next=False):
212         """Event 5"""
213         print("Tower TMU - Event 5")
214     def monitor_traffic_flow(self, duration=-1):
215         """Event 6"""
216         print("Tower TMU - Event 6 continuous")
217     def monitor_and_adjust_flow(self, duration=-1):
218         """Event 7"""
219         print("Tower TMU - Event 7 continuous")
220 class ATCSupervisor(Agent):
221     """ATC Supervisor"""
222     def __init__(self, model, unique_id):
223         self.model = model
224         self.unique_id = unique_id
225         self.doing = dict()
226     def step(self):
```

```
227         iterateoverthis = copy(self.doing)
228         for action, time_remaining in iterateoverthis.items():
229             if time_remaining:
230                 action(duration=self.doing[action])
231                 self.doing[action] -= 1
232             else:
233                 action(duration=self.doing[action], next=True)
234                 self.doing.pop(action)
235     def request(self, function):
236         self.doing[function]=function.__defaults__[0]-1
237     def monitor_and_adjust_flow(self, duration=-1):
238         """Event 7"""
239         print("ATC Supervisor - Event 7 continuous")
240     def coordinate_with_sectors(self, duration=10, next=False):
241         """Event 10"""
242         if duration == 9:
243             for agent in self.model.schedule.agents:
244                 if isinstance(agent, ARTCCTMU):
245                     agent.request(agent.coordinate_ground_stop_and_departure_delay)
246         print("ATC Supervisor - Event 10")
247 class FAAAsIsModel(Model):
248     """The model"""
249     def __init__(self):
250         self.schedule = RandomActivation(self)
251         self.create_agents()
252     def create_agents(self):
253         """Creating the required agents"""
254         self.schedule.add(ATCSCC(self,0))
255         self.schedule.add(SystemUserCommunity(self,1))
256         self.schedule.add(ARTCCTMU(self,2))
257         self.schedule.add(TRACONTMU(self,3))
258         self.schedule.add(TowerTMU(self,4))
259         self.schedule.add(ATCSupervisor(self,5))
260     def step(self, stepno):
261         print("model step " + str(stepno) + ".")
262         self.schedule.step()
263     def run_model(self, steps = 95):
264         for stepno in range(steps):
265             self.step(stepno)
266 if __name__ == '__main__':
267     model = FAAAsIsModel()
```

```
268 model.run_model()
```

Listing B.3: Agent-based simulation for the FAA example using the Mesa Framework in Python

REFERENCES

[1] "IEEE Standard Glossary of Software Engineering Terminology," Dec 1990.

[2] "Integration Definition for Function Modeling (IDEF0)," December 1993.

[3] "DoD modeling and simulation (M&S) glossary," January 1998.

[4] *The American Heritage dictionary of the English language*. Boston: Houghton Mifflin, 2000.

[5] "Unified modeling language specification," January 2005.

[6] "DoDAF product development questionnaire analysis report and new product recommendations report," tech. rep., U.S. Department of Defense, Arlington, VA, May 2008.

[7] "Systems and software engineering system life cycle processes," *ISO/IEC 15288:2008(E) IEEE Std 15288-2008 (Revision of IEEE Std 15288-2004)*, pp. 1–84, 2008.

[8] "OMG Systems Modeling Language," June 2012.

[9] "NATO glossary of terms and definitions," April 2013.

[10] "NATO Architecture Framework v4.0 Documentation," 2018.

[11] 111th Congress, "Public law 111 - 23 - weapon systems acquisition reform," May 2009. Bill Number S. 454, H. Rept. 111-124, H. Rept. 111-101.

[12] AbuSharekh, A., Kansal, S., Zaidi, A. K., and Levis, A. H., "Modeling time in DoDAF compliant executable architectures," in *Conference on Systems Engineering Research*, 2007.

[13] Achinstein, P., *Concepts of Science: A Philosophical Analysis*, ch. 8: On a semantical theory of models, p. 228. Baltimore: Johns Hopkins Press, 1968.

[14] Achinstein, P., *Concepts of Science: A Philosophical Analysis*, ch. 8: On a semantical theory of models, pp. 230–258. Baltimore: Johns Hopkins Press, 1968.

[15] Achinstein, P., *Concepts of Science: A Philosophical Analysis*, ch. 7: Analogies and Models, pp. 209–221. Baltimore: Johns Hopkins Press, 1968.

[16] Air and Land Forces Subcommittee, Committee on Armed Services, House of Representatives, "Air Force procurement: Aerial refueling tanker protest," No. GAO-08-991T, United States Government Accountability Office, July 2008.

[17] Al-Shukri, S., Lenin, R. B., Ramaswamy, S., Imperiale, A., and Itmi, M., "A system of systems approach to modeling and analysis of P2P overlays for WSNs," in *System of Systems Engineering, 2009. SoSE 2009. IEEE International Conference on*, pp. 1–6, 2009.

[18] Angerhofer, B. and Angelides, M., "System dynamics modelling in supply chain management: research review," in *Simulation Conference, 2000. Proceedings. Winter*, vol. 1, pp. 342–351 vol.1, 2000.

[19] ATO Architecture Framework Working Group, "National airspace system enterprise architecture framework," September 2007.

[20] Bagdatli, B. and Mavris, D., "Use of high-level architecture discrete event simulation in a system of systems design," in *Aerospace Conference, 2012 IEEE*, pp. 1–13, March 2012.

[21] Bahill, A. T. and Gissing, B., "Re-evaluating systems engineering concepts using systems thinking," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 28, pp. 516–527, Nov 1998.

[22] Balci, O., "Quality assessment, verification, and validation of modeling and simulation applications," in *Simulation Conference, 2004. Proceedings of the 2004 Winter*, vol. 1, pp. 122–129, 2004.

[23] Balci, O., "Verification, validation and accreditation of simulation models," in *Simulation Conference, 1997., Proceedings of the 1997 Winter*, pp. 135 –141, dec 1997.

[24] Balestrini Robinson, S., *A modeling process to understand complex system architectures*. Dissertation, Georgia Institute of Technology, August 2009.

[25] Barnhart, E., Lei, T., and Ventura, R., "Integrated mission models and simulation through the entire program lifecycle," in *MILITARY COMMUNICATIONS CONFERENCE, 2010 - MILCOM 2010*, pp. 668–673, Oct 2010.

[26] Barnhart, R., *The American Heritage dictionary of science*. Boston: Houghton Mifflin, 1986.

[27] Bauchau, O. A., *Flexible Multibody Dynamics*, vol. 176 of *Solid Mechanics and Its Applications*, ch. 8: Variational and energy principles, pp. 295–299. New York: Springer, 2011. ISBN 978-94-007-0334-6.

[28] Baumgarten, E. and Silverman, S., "Dynamic dodaf power tools," in *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pp. 1–6, Nov 2008.

[29] Belay, A., Helo, P., and Welo, T., "System of systems thinking in product development: A system dynamic approach," in *System of Systems Engineering (SoSE), 2012 7th International Conference on*, pp. 549–554, 2012.

[30] Berman, A., *Nonnegative matrices in the mathematical sciences*, ch. 2: nonnegative matrices, pp. 26–62. Philadelphia: Society for Industrial and Applied Mathematics, 1994.

[31] Berry, D. A., *Bandit problems : sequential allocation of experiments*. Monographs on statistics and applied probability (Series), London ; New York: Chapman and Hall, 1984.

[32] Billington, J., *Application of Petri nets to communication networks : advances in Petri nets*. Berlin New York: Springer, 1999.

[33] Birkler, J., Arena, M. V., Blickstein, I., Drezner, J. A., Gates, S. M., Huang, M., Murphy, R., Nemfakos, C., and Woodward, S. K., *From Marginal Adjustments to Meaningful Change: Rethinking Weapon System Acquisition*, vol. MG-1020-OSD of *Monographs*. Santa Monica, CA: RAND Corporation, 2010.

[34] Boehm, B. W., *Software engineering economics*, ch. 4: The software lifecycle phases and activities, pp. 35–56. Advances in Computing Science and Technology Series, Englewood Cliffs, N.J: Prentice-Hall, 1981.

[35] Bohr, N., "I. On the constitution of atoms and molecules," *Philosophical Magazine Series 6*, vol. 26, no. 151, pp. 1–25, 1913.

[36] Bonabeau, E., "Agent-based modeling: Methods and techniques for simulating human systems," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. Suppl 3, pp. 7280–7287, 2002.

[37] Bowen, R. and Sahin, F., "A system of systems approach to model an artificial immune system using discrete event specification," in *System of Systems Engineering, 2009. SoSE 2009. IEEE International Conference on*, pp. 1–6, 2009.

[38] Box, G., *Empirical model-building and response surfaces*, ch. 3: Least squares for response surface work, p. 74. Probability and mathematical statistics, New York: Wiley, 1987.

[39] Brown, E., Calder, A., Plewa, T., Ricker, P., Robinson, K., and Gallagher, J., "Type Ia supernovae: Simulations and nucleosynthesis," *Nuclear Physics A*, vol. 758, pp. 451–454, July 2005. Nuclei in the Cosmos VIII—Proceedings of the Eighth International Symposium on Nuclei in the Cosmos.

[40] Brown, R., "Calendar queues: a fast 0(1) priority queue implementation for the simulation event set problem," *Commun. ACM*, vol. 31, pp. 1220–1227, Oct. 1988.

[41] Broy, M., Gleirscher, M., Kluge, P., Krenzer, W., Merenda, S., and Wild, D., "Automotive Architecture Framework: Towards a Holistic and Standardised System Architecture Description," Tech. Rep. TUM-I0915, July 2009.

[42] Burbank, J., Kasch, W., and Ward, J., *An Introduction to Network Modeling and Simulation for the Practicing Engineer*, ch. Introduction, pp. 1–19. Hoboken, New Jersey: John Wiley & Sons, Inc., 1st ed., 2011.

[43] Bush, B., Duffy, M., Sandor, D., and Peterson, S., "Using system dynamics to model the transition to biofuels in the united states," in *System of Systems Engineering, 2008. SoSE '08. IEEE International Conference on*, pp. 1–6, 2008.

[44] Caballini, C., Sacone, S., and Siri, S., "The port as a system of systems: A system dynamics simulation approach," in *System of Systems Engineering (SoSE), 2012 7th International Conference on*, pp. 191–196, 2012.

[45] Carnap, R., *Introduction to semantics and formalization of logic*, vol. 1, ch. Relations between semantics and syntax, p. 203. Cambridge, Massachusetts: Harvard University Press, 1943.

[46] Carson, J. S., "Convincing users of model's validity is challenging aspect of modeler's job," *Industrial Engineering*, vol. 18, pp. 74–85, June 1986.

[47] Chairman of the Joint Chiefs of Staff, "CJCSI 3121.01B standing rules of engagement/standing rules for the use of force for U.S. forces," June 2005.

[48] Charette, R., "What's wrong with weapons acquisitions?," *Spectrum, IEEE*, vol. 45, no. 11, pp. 33–39, November.

[49] Congressional Committees, "KC-46 tanker aircraft: Acquisition plans have good features but contain schedule risk," No. GAO-12-366, United States Government Accountability Office, March 2012.

[50] Cook, J., "System of systems reliability for multi-state systems," in *Reliability and Maintainability Symposium, 2009. RAMS 2009. Annual*, pp. 13–18, 2009.

[51] Coppola, A., "FY13 RWDC state aviation challenge." Electronic, 2012.

[52] Dahmann, J., Rebovich, G., Lowry, R., Lane, J., and Baldwin, K., "An implementers' view of systems engineering for systems of systems," in *Systems Conference (SysCon), 2011 IEEE International*, pp. 212–217, April 2011.

[53] Dahmann, J. and Baldwin, K., "Understanding the current state of us defense systems of systems and the implications for systems engineering," in *Systems Conference, 2008 2nd Annual IEEE*, pp. 1–7, 2008.

[54] DARPA Tactical Technology Office, "Meta." Broad Agency Announcement, December 2009. DARPA-BAA-10-21.

[55] Darwin, C. R., *The descent of man, and selection in relation to sex*, vol. 1, ch. 5. On the Development of the Intellectual and Moral Faculties During Primeval and Civilised Times., pp. 158–184. London: John Murray, 1st ed., 1871.

[56] David J. McGill, W. W. K., *Engineering Mechanics: An Introduction to Dynamics, 4th Edition*, ch. 2: Kinetics of Particles and of Mass Centers of Bodies, pp. 56–60. Bloomington, IN: Tichenor Publishing, 2003.

[57] Davis, P., *Generalizing concepts and methods of verification, validation, and accreditation (VV&A) for military simulations*. No. R-4249-ACQ, Santa Monica, CA: RAND Corporation, 1992.

[58] Defense Acquisition University, *Defense Acquisition Guidebook*, ch. 4.1.4. System of Systems (SoS) Engineering, p. 178. Defense Acquisition University, July 2011.

[59] Department of Defense Chief Information Officer, "The DoDAF Architecture Framework Version 2.02," 08 2010.

[60] Department of the Navy, "Marine corps warfighting publication 3-22.2 suppression of enemy air defenses," May 2001.

[61] Dictionary.com, "model," May 2013.

[62] Director for Joint Force Development (J-7), *The Joint Publication 1-02, Department of Defense Dictionary of Military and Associated Terms*. Department of Defense, August 2011.

[63] Dolnick, E., *The clockwork universe : Isaac Newton, the Royal Society, and the birth of the modern world*. New York, NY: HarperCollins, 2011.

[64] Domerçant, J. C., *ARC-VM: an architecture real options complexity-based valuation methodology for military systems-of-systems acquisitions*. Dissertation, Georgia Institute of Technology, November 2011.

[65] Donnelly, R., "Transporting data over radio networks handling rf loss and distinct application needs in a system of systems," in *Systems Conference, 2010 4th Annual IEEE*, pp. 249–254, 2010.

[66] Engler, William O., I., *A methodology for creating expert-based quantitative models for early phase design*. Dissertation, Georgia Institute of Technology, April 2013.

[67] Engler, W., Biltgen, P., and Mavris, D., "Concept selection using an interactive reconfigurable matrix of alternatives (IRMA)," in *45th AIAA Aerospace Sciences Meeting and Exhibit*, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, January 2007.

[68] Fabrycky, W. J. and Blanchard, B. S., *Life-cycle cost and economic analysis*, ch. 1: System life-cycle concepts, pp. 1–14. International series in industrial and systems engineering, Englewood Cliffs, N.J: Prentice Hall, 1991.

[69] Fernandes, R., Li, B., Benjamin, P., and Mayer, R., "Collaboration support for executable enterprise architectures," in *Collaborative Technologies and Systems, 2009. CTS '09. International Symposium on*, pp. 520–527, 2009.

[70] Ford, L. R. and Fulkerson, D. R., *Flows in networks*. Santa Monica, CA: RAND, 1962. http://www.rand.org/pubs/reports/R375.html.

[71] Forrester, J. W., "Industrial dynamics.," *Harvard Business Review*, vol. 36, no. 4, pp. 37–66, 1958.

[72] Forrester, J. W., *Industrial dynamics*, ch. 6: Structure of a Dynamic System Model, pp. 67–72. Cambridge: M. I. T. Press, 1961.

[73] Forrester, J. W., *Industrial dynamics*, ch. 8: Symbols for Flow Diagrams, pp. 81–85. Cambridge: M. I. T. Press, 1961.

[74] Frohberg, K., *Progress in modelling and simulation*, ch. 5: The international linkage of open exchange models, pp. 77–84. London New York: Academic Press, 1982.

[75] Fry, D. and DeLaurentis, D., "Measuring net-centricity," in *System of Systems Engineering (SoSE), 2011 6th International Conference on*, pp. 264–269, 2011.

[76] Galilei, G., *Dialogues concerning two new sciences*, ch. 1: First new science, treating of the resistance which solid bodies offer to fracture. First Day, pp. 66–67. New York: The McMillan Company, 1914. Translated from the Italian and Latin into English by Henry Crew and Alfonso de Salvio. Orig. 1638.

[77] Garrett, R. K., Anderson, S., Baron, N. T., and Moreland, J. D., "Managing the interstitials, a system of systems framework suited for the ballistic missile defense system," *Systems Engineering*, vol. 14, no. 1, pp. 87–109, 2011.

[78] Gass, S., *Decision making, models and algorithms : a first course*, ch. 2: Systems, Models, and Algorithms, pp. 9–22. Malabar, Fla: Krieger Pub. Co, 1991.

[79] Ge, B., Hipel, K., Yang, K., and Chen, Y., "A novel executable modeling approach for system-of-systems architecture," *Systems Journal, IEEE*, vol. PP, no. 99, pp. 1–10, 2013.

[80] Ge, B., Hipel, K. W., Yang, K., and Chen, Y., "A data-centric capability-focused approach for system-of-systems architecture modeling and analysis," *Systems Engineering*, vol. 16, no. 3, pp. 363–377, 2013.

[81] Gelernter, D., *This Will Make You Smarter: New Scientific Concepts to Improve Your Thinking*, ch. Recursive structure, pp. 246–249. New York: HarperCollins, 2012.

[82] Gibson, M. C., Patel, A. B., Nagpal, R., and Perrimon, N., "The emergence of geometric order in proliferating metazoan epithelia," *Nature*, vol. 442, pp. 1038–1041, Aug 2006.

[83] Give Team, "Insight maker." http://insightmaker.com/, 2012.

[84] Gortney, W. E., "Joint capabilities integration and development system." Instruction, January 2012.

[85] Gortney, W. E., "Net ready key performance parameter," March 2012. CJCSI 6212.01F.

[86] Gove, P., *Webster's third new international dictionary of the English language unabridged*. Springfield, Mass., U.S.A: Merriam-Webster, 1993.

[87] Gove, P., *Webster's third new international dictionary of the English language unabridged*. Springfield, Mass., U.S.A: Merriam-Webster, 1993.

[88] Griendling, K. and Mavris, D., "Development of a DoDAF-based executable architecting approach to analyze system-of-systems alternatives," in *Aerospace Conference, 2011 IEEE*, pp. 1–15, 2011.

[89] Griendling, K. and Mavris, D., "A process for systems of systems architecting," in *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, Aerospace Sciences Meetings, (Orlando, FL), American Institute of Aeronautics and Astronautics, January 2010.

[90] Griendling, K. A., *ARCHITECT: the architecture-based technology evaluation and capability tradeoff method*. Dissertation, Georgia Institute of Technology, November 2011.

[91] Grogan, P. and de Weck, O., "An integrated modeling framework for infrastructure system-of-systems simulation," in *Systems Conference (SysCon), 2013 IEEE International*, pp. 483–490, 2013.

[92] Harchol-Balter, M., *Performance Modeling and Design of Computer Systems*. West Nyack, NY, USA: Cambridge University Press, 2013.

[93] Haskins, C., ed., *Systems Engineering Handbook: A Guide For System Life Cycle Processes And Activities*. International Council on Systems Engineering, June 2006.

[94] Hazelrigg, G., *Systems engineering : an approach to information-based design*. Upper Saddle River, NJ: Prentice Hall, 1996.

[95] Hempel, C., *Philosophy of natural science*, ch. 2: Scientific inquiry: invention and test, pp. 6–18. Englewood Cliffs, N.J: Prentice-Hall, 1966.

[96] Hertz, H., *Die Prinzipien der Mechanik in neuem Zusammenhange dargestellt*, ch. Einleitung, p. 1. Gesammelte werke, J. A. Barth, 1894.

[97] Hesse, M. B., *Models and analogies in science*, ch. 1: The function of models: a dialogue, pp. 8–21. Sheed and Ward, 1966.

[98] Hetzel, W., *The complete guide to software testing*. Wellesley, Massachusetts: QED Information Sciences, Inc., 1984.

[99] Hill, R. R., Carl, R. G., and Champagne, L. E., "Using agent-based simulation to empirically examine search theory using a historical case study," *Journal of Simulation*, vol. 1, pp. 29–38, 12 2006.

[100] Hoffman, F. O. and Hammonds, J. S., "Propagation of uncertainty in risk assessments: The need to distinguish between uncertainty due to lack of knowledge and uncertainty due to variability," *Risk Analysis*, vol. 14, no. 5, pp. 707–712, 1994.

[101] Holland, J., *Emergence: from chaos to order*, ch. 1: Before we proceed, pp. 1–15. Reading, Mass: Addison-Wesley, 1998.

[102] Hosking, M. and Sahin, F., "An XML based system of systems agent-in-the-loop simulation framework using discrete event simulation," in *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pp. 3293–3298, 2009.

[103] Hurlburt, G., "Development of the warfighting architecture requirements (war) tool," in *Object-Oriented Real-Time Dependable Systems, 2005. WORDS 2005. 10th IEEE International Workshop on*, pp. 97–104, Feb 2005.

[104] Iacobucci, J. V., *Rapid Architecture Alternative Modeling (RAAM): a framework for capability-based analysis of system of systems architectures*. Dissertation, Georgia Institute of Technology, June 2012.

[105] Information Coherence Authority for Defence, "Ministry of Defense Architecture Framework," December 2012.

[106] International Atomic Energy Agency, *Evaluating the reliability of predictions made using environmental transfer models*, ch. 2: General aspects of model reliability evaluation, pp. 10–12. Vienna: International Atomic Energy Agency, 1989.

[107] ISEE Systems, Inc., "STELLA, Systems Thinking for Education and Research." http://www.iseesystems.com/, 2012.

[108] Jian, X., Bing-feng, G., Xiao-ke, Z., Ke-wei, Y., and Ying-Wu, C., "Evaluation method of system-of-systems architecture using knowledge-based executable model," in *Management Science and Engineering (ICMSE), 2010 International Conference on*, pp. 141–147, Nov 2010.

[109] Jiang, S.-W., Lv, W.-M., and Feng, J.-C., "Research on process modeling and analyzing methods of distributed equipment system-of-systems," in *Quality, Reliability, Risk, Maintenance, and Safety Engineering (ICQR2MSE), 2012 International Conference on*, pp. 161–164, 2012.

[110] Jones-Wyatt, E., Domercant, J., and Mavris, D., "A reliability-based measurement of interoperability for systems of systems," in *Systems Conference (SysCon), 2013 IEEE International*, pp. 408–413, April 2013.

[111] Jones Wyatt, E. A., *A reliability-based measurement of interoperability for conceptual-level systems of systems*. Dissertation, Georgia Institute of Technology, August 2014.

[112] Jonsson, P., Cox, T. J., Primack, J. R., and Somerville, R. S., "Simulations of dust in interacting galaxies. I. dust attenuation," *The Astrophysical Journal*, vol. 637, no. 1, p. 255, 2006.

[113] Kazil, J. and Masad, D., "Mesa: Agent-based modeling in Python 3+." https://github.com/projectmesa/mesa, Sep 2014. Accessed: Sep 2016.

[114] Kendall, D. G., "Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov chain," *The Annals of Mathematical Statistics*, vol. 24, no. 3, pp. 338–354, 1953.

[115] Khalil, W., Merzouki, R., Ould-Bouamama, B., and Haffaf, H., "Hypergraph models for system of systems supervision design," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 42, no. 4, pp. 1005–1012, 2012.

[116] Kilicay-Ergin, N. and Dagli, C., "Executable modeling for system of systems architecting: An artificial life framework," in *Systems Conference, 2008 2nd Annual IEEE*, pp. 1–5, April 2008.

[117] Krygiel, A., *Behind the Wizard's curtain an integration environment for a system of systems*, ch. 2: Systems of systems and federations of systems, pp. 31–47. Washington, D.C: National Defense University, 1999.

[118] Law, A. M. and Kelton, W. D., *Simulation modeling and analysis*, ch. 1: Basic simulation modeling, pp. 1–7. Industrial engineering and management science, New York: McGraw-Hill, 1991.

[119] Law, A. M. and Kelton, W. D., *Simulation modeling and analysis*, ch. 5: Building valid and credible simulation models, pp. 298–324. Industrial engineering and management science, New York: McGraw-Hill, 1991.

[120] Law, A. M. and Kelton, W. D., *Simulation modeling and analysis*, ch. 1: Basic simulation modeling, pp. 106–109. Industrial engineering and management science, New York: McGraw-Hill, 1991.

[121] Law, A. M. and Kelton, W. D., *Simulation modeling and analysis*, ch. 1: Basic simulation modeling, pp. 7–13. Industrial engineering and management science, New York: McGraw-Hill, 1991.

[122] Levitt, M. and Warshel, A., "Computer simulation of protein folding," *Nature*, vol. 253, pp. 694–698, February 1975.

[123] Levy, S., "Google's Larry Page on why moon shots matter," *Wired Magazine*, vol. 21, February 2013.

[124] Li, L., Dou, Y., Ge, B., Yang, K., and Chen, Y., "Executable system-of-systems architecting based on DoDAF meta-model," in *System of Systems Engineering (SoSE), 2012 7th International Conference on*, pp. 362–367, 2012.

[125] Liu, J., Priyantha, B., Hart, T., Ramos, H. S., Loureiro, A. A. F., and Wang, Q., "Energy efficient GPS sensing with cloud offloading," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, (New York, NY, USA), pp. 85–98, ACM, 2012.

[126] Luna-Reyes, L. F. and Andersen, D. L., "Collecting and analyzing qualitative data for system dynamics: methods and models," *System Dynamics Review*, vol. 19, no. 4, pp. 271–296, 2003.

[127] Macal, C. M. and North, M. J., "Tutorial on agent-based modelling and simulation," *Journal of Simulation*, vol. 4, pp. 151–162, September 2010.

[128] Mach, E., *The Science of Mechanics: A Critical and Historical Account of Its Development*, ch. 1: The development of the principles of statics, pp. 40–48. Open court publishing Company, 1919.

[129] Mackay, A. L., *A Dictionary of scientific quotations*. Bristol Philadelphia: A. Hilger, 1991.

[130] Maier, M. W., "Architecting principles for systems-of-systems," *Systems Engineering*, vol. 1, no. 4, pp. 267–284, 1998.

[131] Mane, M., DeLaurentis, D., and Frazho, A., "A Markov perspective on system-of-systems complexity," in *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*, pp. 1238–1243, 2011.

[132] Masad, D. and Kazil, J., "Mesa: An Agent-Based Modeling Framework," in *Proceedings of the 14th Python in Science Conference* (Kathryn Huff and James Bergstra, eds.), pp. 53–60, 2015.

[133] Math Works, "Ordinary differential equations."

[134] Mathieu, J. and Callaway, D., "State estimation and control of heterogeneous thermostatically controlled loads for load following," in *System Science (HICSS), 2012 45th Hawaii International Conference on*, pp. 2002–2011, Jan 2012.

[135] Mavris, D., DeLaurentis, D., Bandte, O., and Hale, M., "A stochastic approach to multi-disciplinary aircraft analysis and design," in *36th AIAA Aerospace Sciences Meeting and Exhibit*, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, January 1998.

[136] Mavris, D. and Griendling, K., "Relational oriented systems engineering and technology tradeoff analysis (ROSETTA) environment," in *System of Systems Engineering (SoSE), 2011 6th International Conference on*, pp. 49–54, 2011.

[137] Mehlhorn, K. and Sanders, P., *Algorithms and data structures : the basic toolbox*, ch. 4: Hash tables and associative arrays, pp. 81–98. Berlin : Springer, c2008., 2008.

[138] Merriam-webster.com, "model," May 2013.

[139] Metz, C., "If Xerox PARC invented the PC, Google invented the internet," August 2012.

[140] Minsky, M. L., *Semantic information processing*, ch. 9: Matter, mind, and models, pp. 425–426. Cambridge, Massachusetts: MIT Press, 1968.

[141] Mittal, S., Mitra, A., Gupta, A., and Zeigler, B., "Strengthening OV-6a semantics with rule-based meta-models in DEVS/DoDAF based life-cycle architectures development," in *Information Reuse and Integration, 2006 IEEE International Conference on*, pp. 80–85, Sept 2006.

[142] Mittal, S., "Extending dodaf to allow integrated devs-based modeling and simulation," *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, vol. 3, no. 2, pp. 95–123, 2006.

[143] Mittal, S., Zeigler, B. P., Martin, J. L. R., Sahin, F., and Jamshidi, M., *System of systems engineering : innovations for the 21st century*, ch. 5: Modeling and simulation for systems of systems engineering, pp. 101–149. Systems engineering and management, Hoboken, N.J: Wiley, 2009.

[144] Morris, C., *Academic Press dictionary of science and technology*. San Diego: Academic Press, 1992.

[145] Morris, C., *Academic Press dictionary of science and technology*. San Diego: Academic Press, 1992.

[146] Murata, T., "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[147] Murphy, G., *Similitude in engineering*, ch. 4: Theory of models, pp. 57–72. Ronald Press Co., 1950.

[148] Myers, R. H. and Montgomery, D. C., *Response surface methodology : process and product optimization using designed experiments*, ch. 2: Building Empirical Models, pp. 41–43. Probability and statistics, New York: J. Wiley, 2nd ed., 2002.

[149] NAS Enterprise Architecture Team, AJP-15, "NAS As Is (2011) Operational Node Connectivity Diagram (OV-2)," September 2011.

[150] National Aeronautics and Space Administration, "Environmentally responsible aviation project." http://www.aeronautics.nasa.gov/iasp/era/index.htm, Aug 2015.

[151] Newton, I., Motte, A., and Machin, J., *The Mathematical Principles of Natural Philosophy*. No. v. 1 in The Mathematical Principles of Natural Philosophy, B. Motte, 1729.

[152] No Magic, "UPDM Plugin," August 2018.

[153] Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Systems and Software Engineering, *Systems Engineering Guide for Systems of Systems, Version 1.0*, ch. 1.4: Definition of Terms, p. 4. Washington, DC: ODUSD(A&T)SSE: Department of Defense, 2008.

[154] on Pre-Milestone A Systems Engineering: A Retrospective Review, C. and Benefits for Future Air Force Systems Acquisition, N. R. C., *Pre-Milestone A and Early-Phase Systems Engineering: A Retrospective Review and Benefits for Future Air Force Acquisition*, ch. 1, pp. 14–25. The National Academies Press, 2008.

[155] on Pre-Milestone A Systems Engineering: A Retrospective Review, C. and Benefits for Future Air Force Systems Acquisition, N. R. C., *Pre-Milestone A and Early-Phase Systems Engineering: A Retrospective Review and Benefits for Future Air Force Acquisition*. The National Academies Press, 2008.

[156] Parker, S., *McGraw-Hill dictionary of scientific and technical terms*. New York: McGraw-Hill, 1994.

[157] Pawlowski, T., Barr, P. C., and Ring, S. J., "Applying executable architectures to support dynamic analysis of C2 systems," tech. rep., The MITRE Corporation, 245 Sedgwick Avenue, Fort Leavenworth, KS, 66027, June 2004.

[158] Peirce, C. S., *Collected Papers of Charles Sanders Peirce*, vol. 1, book I, ch. 2: Lessons from the history of science, p. 20. Cambridge, Massachusetts: Harvard University Press, 1931. paragraph 46.

[159] Peirce, C. S., *Collected Papers of Charles Sanders Peirce*, vol. 2, book III, ch. 5: Ampliative reasoning, p. 385. Cambridge, Massachusetts: Harvard University Press, 1931. paragraph 640.

[160] Peirce, C. S., *Collected Papers of Charles Sanders Peirce*, vol. 2, book III, ch. 8: A theory of probable inference, pp. 441–443. Cambridge, Massachusetts: Harvard University Press, 1931. paragraphs 702–703.

[161] Planck, M., "Ueber das gesetz der energieverteilung im normalspectrum," *Annalen der Physik*, vol. 309, no. 3, pp. 553–563, 1901.

[162] Powersim Software AS, "Powersim." http://www.powersim.com/, 2012.

[163] Prandtl, L., "Tragflügeltheorie. erste mitteilung," *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, pp. 451–477, 1918.

[164] Proth, *Petri nets : a tool for design and management of manufacturing systems*. Chichester New York: Wiley, 1996.

[165] Reddy, V. N., "Modeling biological pathways: a discrete event systems approach," Master's thesis, University of Maryland, 1994. Series ISR, MS 1994-4.

[166] Regli, W., Mayk, I., Dugan, C., Kopena, J., Lass, R., Modi, P., Mongan, W., Salvage, J., and Sultanik, E., "Development and specification of a reference model for agent-based systems," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 39, no. 5, pp. 572–596, 2009.

[167] Robinson, S., *Simulation: the practice of model development and use*, ch. 1: www.simulation: what, why, and when?, pp. 1–12. Chichester, England: John Wiley & Sons, Ltd., 2004.

[168] Robinson, S., *Simulation: the practice of model development and use*, ch. 12: Verification, validation, and confidence, pp. 209–225. Chichester, England: John Wiley & Sons, Ltd., 2004.

[169] Robinson, S., *Simulation: the practice of model development and use*, ch. 6: Conceptual modelling, pp. 63–94. Chichester, England: John Wiley & Sons, Ltd., 2004.

[170] Rodrigues, R., "Modeling a real case metasystem architecture using finite state process formalism," in *System of Systems Engineering (SoSE), 2013 8th International Conference on*, pp. 279–284, June 2013.

[171] Sage, A., *Software systems engineering*, ch. 5: Design and Evaluation of Decision Support Systems, pp. 161–204. New York: Wiley, 1990.

[172] Sanchez-Acevedo, M., Lopez-Mellado, E., and Ramos-Corchado, F., "Mobile agents formation control in 3D environments based on self organization strategies," in *System of Systems Engineering, 2009. SoSE 2009. IEEE International Conference on*, pp. 1–6, 2009.

[173] Schlett, M., "Trends in embedded-microprocessor design," *Computer*, vol. 31, no. 8, pp. 44–49, 1998.

[174] Schrage, D. P. and Mavris, D. N., "Integrated design and manufacturing for the high speed civil transport," in *Aircraft Design, Systems and Operations Meeting*, AIAA-1993-3994, (Monterey, CA), Aug 11-13 1993.

[175] Shafer, A., Benjamin, M., Leonard, J., and Curcio, J., "Autonomous cooperation of heterogeneous platforms for sea-based search tasks," in *OCEANS 2008*, pp. 1–10, Sept 2008.

[176] Simpson, J., *The Oxford English dictionary*. Oxford: Clarendon Press, 1989.

[177] Software Engineering Committee of the IEEE Computer Society, "IEEE guide for developing system requirements specifications," *IEEE Std 1233, 1998 Edition*, pp. 1–36, 1998.

[178] Sterman, J. D., "System dynamics modeling: Tools for learning in a complex world.," *California Management Review*, vol. 43, no. 4, pp. 8–25, 2001.

[179] Suh, N., *The principles of design*. New York: Oxford University Press, 1990.

[180] Tawada, Y., *Das Fremde aus der Dose*. Graz: Literaturverlag Droschl, 1992.

[181] Thayer, R. H., Fairley, R. E., and Bjorke, P., "IEEE guide for information technology - system definition - concept of operations (conops) document," *IEEE Std 1362-1998*, pp. 1–24, 1998.

[182] The Object Management Group, "Semantics of a foundational subset for executable uml models," October 2017.

[183] TheFreeDictionary.com, "model," May 2013.

[184] Tietjens, O. G., *Applied Hydro- and Aerodynamics*, ch. 4: Airfoil theory, p. 207. Engineering Societies Monographs, New York: McGraw-Hill Book Company, Inc., 1st ed., 1934. Based on lectures of L. Prandtl and translated by J. P. Den Hartog.

[185] Turner, A. J., *A methodology for the development of models for the simulation of non-observable systems*. Dissertation, Georgia Institute of Technology, 2014.

[186] US Joint Forces Command, *The Joint Publication 4-09, Distribution Operations*. Department of Defense, December 2013.

[187] Ventana Systems, Inc., "Vensim." http://vensim.com/, 2012.

[188] Volovoi, V., "Modeling of system reliability petri nets with aging tokens," *Reliability Engineering & System Safety*, vol. 84, no. 2, pp. 149–161, 2004.

[189] von Bortkiewicz, L., *Das Gesetz der kleinen Zahlen*, ch. 2: Anwendung der Formeln des 1. Kapitels auf einige Daten der Selbstmord- und der Unfall-Statistik, pp. 17–25. B.G. Teubner, 1898.

[190] Wagenhals, L. W., Haider, S., and Levis, A. H., "Synthesizing executable models of object oriented architectures," in *Proceedings of the Conference on Application and Theory of Petri Nets: Formal Methods in Software Engineering and Defence Systems - Volume 12*, CRPIT '02, (Darlinghurst, Australia, Australia), pp. 85–93, Australian Computer Society, Inc., 2002.

[191] Walpole, R. E., Myers, R. H., Myers, S. L., and Ye, K., *Probability & statistics for engineers & scientists*, ch. 6: Some continuous probability distributions, pp. 171–209. Upper Saddle River, NJ: Pearson Prentice Hall, 2007.

[192] Wang, R. and Dagli, C., "An executable system architecture approach to discrete events system modeling using sysml in conjunction with colored petri net," in *Systems Conference, 2008 2nd Annual IEEE*, pp. 1 –8, april 2008.

[193] Wasson, C., *System analysis, design, and development concepts, principles, and practices*. Hoboken, N.J: Wiley-Interscience, 2006.

[194] Weisstein, E. W., "Bipartite graph."

[195] Wennergren, D. M., "The Department of Defense Architecture Framework Version 2.0." Memorandum, May 2009.

[196] Whittaker, E. T. Sir and Robinson, G., *The calculus of observations : a treatise on numerical mathematics*, ch. 12: Correlation, pp. 317–342. London: Blackie and Son, Limited, 1924.

[197] Wilensky, U., "NetLogo." http://ccl.northwestern.edu/netlogo/, 1999.

[198] Wolstenholme, E. F. and Coyle, R. G., "The development of system dynamics as a methodology for system description and qualitative analysis," *The Journal of the Operational Research Society*, vol. 34, no. 7, pp. pp. 569–581, 1983.

[199] Xiao-li, B., Xue-shan, L., Xiao-hui, B., Xian-qing, Y., Hong-hui, C., and De-ke, G., "Study of dod architecture simulation validation based on uml and extended colored petri nets," in *Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference on*, pp. 61–66, April 2008.

[200] XJ Technologies Company, "Anylogic," 2012.

[201] Xu, D. and Deng, Y., "Modeling mobile agent systems with high level petri nets," in *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, vol. 5, pp. 3177–3182 vol.5, 2000.

[202] Youssef, R., Kim, B., Pagotto, J., Vallerand, A., Lam, S., Pace, P., Pogue, C., and Greenley, A., "Toward an integrated executable architecture and M&S based analysis for counter terrorism and homeland security," in *Transforming Training and Experimentation through Modelling and Simulation*, no. RTO-MP-MSG-045, (Neuilly-sur-Seine, France), pp. 7–1–7–24, 2006.

[203] Zachman, J. A., "A framework for information systems architecture," *IBM Systems Journal*, vol. 38, no. 2, pp. 454–470, 1999.

[204] Zinn, A. W., "The use of integrated architecture to support agent based simulation: an initial investigation," Master's thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, March 2004. AFIT/GSE/ENY/04-M01.

VITA

Burak Bagdatli received his B.S., M.S., and Ph.D. degrees in Aerospace Engineering from Georgia Institute of Technology. His primary research area is system of systems architectures and simulations. He worked as a design engineer on a medium altitude high endurance UAV at Turkish Aerospace Industries. Burak also worked on several research projects including modeling and analysis of new technologies, concepts, and operations for the Next Generation Air Transportation System, coding simulations for architecture-based technology evaluation and capability tradeoffs, developing an interactive web-based course on electromechanical systems, analyzing, visualizing, and working with large-scale manufacturing data, creation of digital twin for manufacturing machines, and exploring design spaces for model-based commercial airline designs. His research interests include discrete event, agent-based, and mathematical simulations, digital twin for various phases of a program, and statistical modeling and visualizations.